

Project 5 Write Up:

Project Description:

In Project 5 we were required to implement a simple form of protection for the xv6 file system using mode bits. This was accomplished through the use of some new additions to the on disk and in memory inode structs as well as the implementation of new user commands and system calls. We learned some new concepts related to the way the file system is implemented, the use of transactions, and the idea of persistence in order to provide xv6 with some measure of protection if faced with some sort of loss of power or reboot.

Project Deliverables:

The following functionality was added to the xv6 operating system:

1. New System Calls
 - Added the chmod system call
Prototype: `int chmod(char *pathname, int mode)`
 - Added the chown system call
Prototype: `int chown(char *pathname, int owner)`
 - Added the chgrp system call
Prototype: `int chgrp(char *pathname, int owner)`
2. New Commands
 - Added the chmod command
Command Example: `chmod STRING STRING`
where each STRING are command-line arguments that the command takes
 - Added the chown command
Command Example: `chown INT STRING`
where int and string are command-line arguments that the command takes
 - Added the chgrp command
Command Example: `chgrp INT STRING`
where int and string are command-line arguments that the command takes
3. Modified System Calls
 - Modified the exec system call
 - Modified the fstat system call
4. Modified Commands
 - Modified the ls command
 - Modified the mkfs command
5. Modified inode/dinode structs
 - Added mode_t union
 - Added uid, gid, and mode to dinode
6. Modified stat struct
 - Added stat_mode_t union

- Added uid, gid, and mode to stat struct
7. Modified iupdate/ilock routines
- Changes described in implementation section #8.

Project Implementation:

The following additions to the xv6 system were written incorporating a new compilation flag CS333_P5, added to the Makefile (lines 76 and 132).

1. New System Calls

Files Modified: usys.S, user.h, syscall.h, syscall.c, sysfile.c, defs.h (lines 54-56)

The following system calls were added to the xv6 operating system using the same methods described and used in prior projects. These kernel-side system calls reside in the sysfile.c file along with other system calls used for the file system. They are used primarily for error checking the values that are grabbed off the stack, passed in via the commands that call them.

- **chmod(void)** - The chmod system call (lines 446-456) takes a void argument and uses both argptr() and argint() to grab the mode and file name off the stack, which were passed into the chmod command that calls it's user-side equivalent that takes those two arguments. If the file name isn't retrieved correctly, a -1 is returned to indicate failure. The mode and file name are then passed into the chmod_helper() function (line 459). The chmod system call returns an integer based on the result of the chmod_helper() function.
- **chown(void)** - The chown system call (lines 463-476) takes a void argument and uses both argptr() and argint to grab the UID and file name off the stack, which were passed into the chown command that calls it's user-side equivalent that takes those two arguments. If the file name isn't retrieved correctly, a -1 is returned to indicate failure. The upper and lower bounds of the UID retrieved from the stack are then checked to make sure that the UID is in the range $0 \leq \text{UID} \leq 32767$. If the UID is not within the appropriate range, a -1 is returned to indicate failure (lines 472-473). Otherwise, the chown system call returns the value of a call to the chown_helper() function, passing in the UID and the file name as arguments (line 475).
- **chgrp(void)** - The chgrp system call (lines 479-492) takes a void argument and uses both argptr() and argint() to grab the GID and file name off the stack, which were passed into the chgrp command that calls it's user-side equivalent that takes those two arguments. If the file name isn't retrieved correctly, a -1 is returned to indicate failure. The upper and lower bounds of the GID retrieved from the stack are then checked to make sure that the GID is in the range $0 \leq \text{GID} \leq 32767$. If the GID is not within the appropriate range, a -1 is returned to indicate failure (lines 488-489). Otherwise, the chgrp system call returns the value of a call to the chgrp_helper() function, passing in the GID and the file name as arguments (line 491).

2. New commands

Files Added: chmod.c, chown.c, chgrp.c

Files Modified: Makefile (), runoff.list (bottom of file)

The following new user commands were added as new files to the xv6 system using the same methods used to add new user commands detailed in prior assignments.

- **chmod(int mode, char *pathname)** - The chmod user command takes in two command line arguments: a mode and a path name for a file. The chmod command first checks against the number of expected arguments to make sure that the appropriate amount of argument (2) are entered (lines 10-12). If an incorrect amount of arguments are entered an error message is printed and the command exits. The mode argument is then captured and the length of the string version of the argument is checked to make sure that it has the appropriate length (4).

If it's not the appropriate length, an error message is printed and the command exits (lines 15-18). The pathname argument is then captured and the length of the argument is checked to make sure that something was actually entered for the file name (lines 21-24). If incorrectly entered, an error message is printed and the command exits. Next, the mode argument is checked in order to ensure that it is in the valid range 0000-1777 (lines 29-44). If it's not in the valid range an error message is printed and the command exits. The mode is then converted from its string octal value to its decimal equivalent (lines 47-50). The arguments are then passed into the user-side version of the chmod system call which invokes the kernel-side version of the chmod system call (line 53). The return code for this call is checked in order to output success or failure before exiting the command.

- **chown(int owner, char *pathname)** – The chown user command takes in two command line arguments: a UID and a path name for a file. First, the user command checks to make sure that the appropriate amount of arguments were entered. If not, an appropriate error message is printed and the command exits (lines 9-12). Next, the command uses the atoi() function to convert the UID argument taken in via the command line from a string to its integer equivalent (line 14). A bounds check is performed to make sure that the integer UID is within the range $0 \leq \text{UID} \leq 32767$. If its not in the correct range, an error message is printed and the command exits (lines 15-18). Next, the command grabs the path name argument from the command line and uses the strlen() function to make sure that the length of the path name is greater than 0. If not, an error message is printed and the command exits (lines 20-24). If all checks pass, the UID and the path name are passed into the user-side version of the chown system call (line 26). If the return value of the system call indicates failure for any reason, the command prints an error message and exits (lines 27-30). Otherwise, the command indicates success by exiting without any error message.
- **chgrp(int owner, char *pathname)** – The chgrp user command takes in two command line arguments: a GID and a path name for a file. First, the user command checks to make sure that the appropriate amount of arguments were entered. If not, an appropriate error message is printed and the command exits (lines 9-12). Next, the command uses the atoi() function to convert the GID argument taken in via the command line from a string to its integer equivalent (line 14). A bounds check is performed to make sure that the integer GID is within the range $0 \leq \text{GID} \leq 32767$. If its not in the correct range, an error message is printed and the command exits (lines 15-18). Next, the command grabs the path name argument from the command line and uses the strlen() function to make sure that the length of the path name is greater than 0. If not, an error message is printed and the command exits (lines 20-24). If all checks pass, the GID and the path name are passed into the user-side version of the chgrp system call (line 26). If the return value of the system call indicates failure for any reason, the command prints an error message and exits (lines 27-30). Otherwise, the command indicates success by exiting without any error message.

3. Modified System Calls

Files Modified: exec.c (lines 31-46 and 114-118), fs.c (lines 439-443)

- **exec()** - The exec system call was modified to incorporate the newly created flag bits in the mode_t union. The exec system call now performs a few checks in order to determine whether a file should be run or not. First, a stat variable is declared and the stati() function is called, passing in the ip inode and the new stat variable (lines 32-33). This copies the file's inode information into the stat variable so that we can have access to the file's inode information solely for comparative purposes. Next, a series of checks is performed in order

to determine whether or not the file can be executed by the currently running process. These checks occur in specific order starting with the process' UID.

- If the UID of the process matches the UID of the file then the user execute bit is checked to see if the file can be executed. If the bit is a 0 it cannot be run and a goto is performed to jump to the bad section of exec. Otherwise the file can be executed by the process using the user bit permissions.
- Next, if the GID of the process matches the GID of the file then the group execute bit is checked to see if the file can be executed. If the bit is a 0 it cannot be executed and a goto is performed to jump to the bad section of exec. Otherwise the file can be executed by the process using the group bit permissions.
- If neither the UID or GID for the process match that of the file, then the other bit is checked to see if the file can be executed. If the bit is a 0 it cannot be executed and a goto is performed to jump to the bad section of exec. Otherwise the file can be executed by the process using the other bit permissions.

Finally, when the new image is committed, if the setuid bit is a 1 then the UID of the process is set to the UID of the file (lines 114-118).

- **fstat()** - The fstat system call was modified by adding the new UID, GID, and mode to the information that gets copied from an inode to a stat variable in the stati() function in fs.c (lines 439-443). This allows the new information to be properly printed to the screen when the ls command is used to display directory and file information.

4. Modified Commands

Files Modified: ls.c (lines 25-86, 122, and), mkfs.c (lines 232-236), param.h

- **ls()** - The print_mode() function was added to the ls.c file (lines 25-86) to handle the formatting for the way the mode bits are printed when the ls command is used. If the ls command is used with a file name given for an argument it prints the column information (line 112), calls print_mode() to print the mode bits for the file (line 113), and then uses a print statement to print the additional file information: name, uid and gid, inode number, and size (line 114). This prints the information for just that one argument given. A print statement was added (line 128) to print out the column name above all the ls output so that it is clear what each column of information represents. Inside the existing while loop in the ls command (lines 130-145) a call to the print_mode() function was added to print the mode bits for each file and a print statement follows that in order to print the remaining relevant file information: the name, the new uid and gid fields, inode number, and size information for the file (lines 142-143).
- **mkfs()** - The ialloc() function was modified so that, upon allocation, on disk nodes uid, gid, and mode bits are set to default values (lines 232-236). The default uid and gid values used for the dinode's are the same uid and gid values used for processes. The default mode value is a new addition to the param.h file for project 5. It's default value is set as 00755 where the leading 0 signifies it is an octal value.

5. Modified inode/dinode structs

Files Modified: fs.h (lines 31-63), file.h (lines 23-27)

- **dinode** – fs.h
 - Another #define for the number of direct inodes, NDIRECT, was added to a conditional compilation statement and set to 10 in fs.h (line 24) in order to incorporate the two new integer fields in the inode.

- A new union `mode_t` was created in order to represent the mode bits for an inode (lines 32-47). This union contains a struct called `flags` and an integer called `as_int` which represents the octal value as an integer. Each digit in the mode corresponds to a specific set of permissions. The first digit of the octal represents the setuid bit, the second digit represents the file permissions for a user, the third digit represents the file permissions for a group, and the fourth digit represents the file permissions for other. The `as_int` variable stores the value of the inode's mode and, since it is a union and the flags share the memory address of the `as_int` variable, when we set `as_int` to a value each individual flag updates to the correct bit.
- Variables for a uid, gid, and the new mode union were added to the `dinode` struct (lines 57-59). These additions incorporate the new fields in the on disk inodes so that, when changes are made to the file's inode we can write these changes to disk so that these changes will persist even after xv6 is shut down and rebooted.
- **Inode – file.h**
Added the same uid, gid, and mode fields to the inode struct (lines 24-26) so that the in memory version of inodes have the same fields as the on disk inodes.

6. Modified stat.h file

Files Modified: stat.h (lines 6-21, and 29-33)

The stat.h file was modified to include its own, identical version of the new `mode_t` union (lines 6-21) from fs.h, renamed to `stat_mode_t` to avoid identically named unions. Using an identical version of the `mode_t` union with a slightly different name is used in this assignment solely to reduce the amount of work required by the student for this assignment. The stat struct was modified (lines 29-33) to include the new uid, gid, and mode variables that are copied from the inode into the stat variable during runtime.

7. New helper functions for system calls

Files Modified: fs.c, defs.h

All function prototypes for new helper functions were added to the defs.h file (lines 55-57). The implementation of these new helper functions is done in fs.c in order to have access to the table of inodes.

- **chown_helper()** - (lines 669-685)

The `chown_helper()` function takes two arguments: a string representing the pathname and an integer representing a UID. First, `begin_op()` is used to start a transaction. The `namei()` function is then used, passing in the pathname argument, to return a pointer to the inode associated with that pathname (line 673). If the pointer returned by `namei()` is null it means that no inode was found associated with the pathname given, and the transaction is ended and a -1 is returned for failure (lines 674-676). Otherwise, an inode was found associated with that pathname. `ilock()` is then used to lock that inode, the uid of that inode is then changed to the uid argument passed into the function, and `iupdate()` is used to copy the changes to disk (lines 678-680). the inode is then unlocked and the transaction is ended. A 0 is returned to indicate a successful change of the uid.

- **chgrp_helper()** - (lines 687-703)

The `chgrp_helper()` function takes two arguments: a string representing the pathname and an integer representing a GID. First, `begin_op()` is used to start a transaction. The `namei()` function is then used, passing in the pathname argument, to return a pointer to the inode associated with that pathname (line 691). If the pointer returned by `namei()` is null it means that no inode was found associated with the pathname given, and the transaction is ended

and a -1 is returned for failure (lines 692-694). Otherwise, an inode was found associated with that pathname. `ilock()` is then used to lock that inode, the uid of that inode is then changed to the uid argument passed into the function, and `iupdate()` is used to copy the changes to disk (lines 696-698). the inode is then unlocked and the transaction is ended. A 0 is returned to indicate a successful change of the gid.

- **`chmod_helper()`** - (lines 705-721)

The `chmod_helper()` function takes two arguments: a string representing the pathname and an integer representing a mode. First `begin_op()` is used to start a transaction. The `namei()` function is then used, passing in the pathname argument, to return a pointer to the inode associated with that pathname (line 709). If the pointer returned by `namei()` is null it means that no inode was found associated with the pathname given, and the transaction is ended and a -1 is returned for failure (lines 710-712). Otherwise, an inode was found associated with that pathname. `ilock()` is then used to lock that inode, the mode of that inode is then changed to the mode argument passed into the function, and `ipupdate()` is used to copy the changes to disk (lines 714-716). The inode is then unlocked and the transaction is ended. A 0 is returned to indicate a successful change of the gid.

8. Modified `iupdate/ilock` routines

Files Modified: `fs.c`

- Modified the `iupdate` routine (lines 213-215) so that the new uid, gid, and mode information is written to disk when changes are made to a file's inode information.
- Modified the `ilock` routine (lines 295-297) so that the new uid, gid, and mode information are also read from the disk when necessary.

9. Added test program

- Added the `p5-test` program to `UPROGS` in the `Makefile` and to the `runoff.list` for testing purposes. All tests are described in the `p5-test` portion of the document at the bottom in full detail.

As a side note, I would like to finish up the implementation section by mentioning that, upon creation of the file system, the default uid and gid values for files are set to the same value as the default uid and gid for processes. This is to avoid problems with changing the permissions for `chmod`, `chown`, and `chgrp`.

Testing:

1. chmod user command test

In order to test that the chmod user command functions properly testing will be split into three sub-tests:

- Valid parameters sub-test

To test whether the chmod user command successfully sets the mode bits of a file I will boot xv6 and use the ls command to output the default information for the files in xv6, including the default mode bits displayed on the leftmost side of the output. I will then run the chmod command passing in a valid mode and a valid file (date) currently in the xv6 ls output. I will follow this up by using the ls command to check the files in xv6 to see if the file used in the chmod command had its mode bits changed appropriately. I expect that, after using the chmod command to change the mode bits for a valid file in xv6, the ls output on the left for that file will have changed appropriately.

```
$ 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	14492
-rwxr-xr-x	echo	0	0	4	13712
-rwxr-xr-x	forktest	0	0	5	9428
-rwxr-xr-x	grep	0	0	6	16136
-rwxr-xr-x	init	0	0	7	14268
-rwxr-xr-x	kill	0	0	8	13784
-rwxr-xr-x	ln	0	0	9	13688
-rwxr-xr-x	ls	0	0	10	17560
-rwxr-xr-x	mkdir	0	0	11	13840
-rwxr-xr-x	rm	0	0	12	13820
-rwxr-xr-x	sh	0	0	13	27520
-rwxr-xr-x	stressfs	0	0	14	14404
-rwxr-xr-x	usertests	0	0	15	59592
-rwxr-xr-x	wc	0	0	16	15004
-rwxr-xr-x	zombie	0	0	17	13488
-rwxr-xr-x	halt	0	0	18	13444
-rwxr-xr-x	date	0	0	19	14976
-rwxr-xr-x	idtest	0	0	20	16948

Figure 1: Valid Parameters Before

```
$ chmod 0255 date
String in octal is 0255 and decimal is 173
$ 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	14492
-rwxr-xr-x	echo	0	0	4	13712
-rwxr-xr-x	forktest	0	0	5	9428
-rwxr-xr-x	grep	0	0	6	16136
-rwxr-xr-x	init	0	0	7	14268
-rwxr-xr-x	kill	0	0	8	13784
-rwxr-xr-x	ln	0	0	9	13688
-rwxr-xr-x	ls	0	0	10	17560
-rwxr-xr-x	mkdir	0	0	11	13840
-rwxr-xr-x	rm	0	0	12	13820
-rwxr-xr-x	sh	0	0	13	27520
-rwxr-xr-x	stressfs	0	0	14	14404
-rwxr-xr-x	usertests	0	0	15	59592
-rwxr-xr-x	wc	0	0	16	15004
-rwxr-xr-x	zombie	0	0	17	13488
-rwxr-xr-x	halt	0	0	18	13444
--w-r-xr-x	date	0	0	19	14976
-rwxr-xr-x	idtest	0	0	20	16948

Figure 2: Valid Parameters After

As we can see in Figure 1 and Figure 2 above, initially the date file shares the same mode values as the other files in the ls output, the default value of 0755. After running the chmod command with a valid mode value of 0255 the date command accurately reflects the mode

changes on the left hand side of the output. 0255 corresponds to 0 010 101 101 for the mode bits which means that: users can't read the file, can write to the file, and can't execute the file; group can read and execute the file but not write to it; and other can read and execute the file but not write to it. This is accurately reflected in the output. The valid parameters sub-test PASSES.

- Invalid filename sub-test

In order to test that the chmod user command doesn't modify anything when given an invalid file name I will use the ls command to output xv6 file information. I will then use the chmod command and give the command a file name that doesn't exist in the xv6 system. I will then use the ls command to output the same file information to check and see if anything has changed in the output. I expect that the chmod command will successfully indicate failure with an appropriate message and that no information will change in the ls output.

```
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     14492
-rwxr-xr-x echo        0         0         4     13712
-rwxr-xr-x forktest    0         0         5      9428
-rwxr-xr-x grep        0         0         6     16136
-rwxr-xr-x init        0         0         7     14268
-rwxr-xr-x kill        0         0         8     13784
-rwxr-xr-x ln          0         0         9     13688
-rwxr-xr-x ls          0         0        10     17560
-rwxr-xr-x mkdir       0         0        11     13840
-rwxr-xr-x rm          0         0        12     13820
-rwxr-xr-x sh          0         0        13     27520
-rwxr-xr-x stressfs    0         0        14     14404
-rwxr-xr-x usertests   0         0        15     59592
-rwxr-xr-x wc          0         0        16     15004
-rwxr-xr-x zombie      0         0        17     13488
-rwxr-xr-x halt        0         0        18     13444
-rwxr-xr-x date        0         0        19     14976
-rwxr-xr-x idtest      0         0        20     16948
-rwxr-xr-x ps          0         0        21     15424
-rwxr-xr-x time        0         0        22     14480
-rwxr-xr-x zombieTest  0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest       0         0        25     14500
-rwxr-xr-x priotest    0         0        26     14360
-rwxr-xr-x chmod       0         0        27     14820
-rwxr-xr-x chown       0         0        28     14112
-rwxr-xr-x chgrp       0         0        29     14112
-rwxr-xr-x ps-test     0         0        30     28556
c----- console      0         0        31         0
```

Figure 3: Invalid file name before

```
$ chmod 0255 helloworld
String in octal is 0255 and decimal is 173
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 cat         0         0         3     14492
-rwxr-xr-x echo        0         0         4     13712
-rwxr-xr-x forktest    0         0         5      9428
-rwxr-xr-x grep        0         0         6     16136
-rwxr-xr-x init        0         0         7     14268
-rwxr-xr-x kill        0         0         8     13784
-rwxr-xr-x ln          0         0         9     13688
-rwxr-xr-x ls          0         0        10     17560
-rwxr-xr-x mkdir       0         0        11     13840
-rwxr-xr-x rm          0         0        12     13820
-rwxr-xr-x sh          0         0        13     27520
-rwxr-xr-x stressfs    0         0        14     14404
-rwxr-xr-x usertests   0         0        15     59592
-rwxr-xr-x wc          0         0        16     15004
-rwxr-xr-x zombie      0         0        17     13488
-rwxr-xr-x halt        0         0        18     13444
-rwxr-xr-x date        0         0        19     14976
-rwxr-xr-x idtest      0         0        20     16948
-rwxr-xr-x ps          0         0        21     15424
-rwxr-xr-x time        0         0        22     14480
-rwxr-xr-x zombieTest  0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest       0         0        25     14500
-rwxr-xr-x priotest    0         0        26     14360
-rwxr-xr-x chmod       0         0        27     14820
-rwxr-xr-x chown       0         0        28     14112
-rwxr-xr-x chgrp       0         0        29     14112
-rwxr-xr-x ps-test     0         0        30     28556
c----- console      0         0        31         0
```

Figure 4: Invalid file name after

As we can see in Figure 3 and Figure 4 above, the output from ls is the exact same both before and after the attempt to give the chmod command an invalid file name. We can also see that the chmod command indicated failure and printed an error message to confirm this outcome. The invalid file name sub-test PASSES.

- Invalid mode sub-test

In order to test that the chmod user command doesn't modify anything when an invalid mode is passed in I will use the ls command to output xv6 file information. I will then use the chmod command and pass in an invalid mode with a valid file name (date) and then use the ls command again to output file information to see if anything has changed. I expect that the chmod user command will successfully indicate failure when an invalid mode is entered and that no changes will happen to any xv6 files.

```
xv6...
cpu0: starting
cpu1: 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        1024
drwxr-xr-x ..     0        0        1        1024
-rwxr-xr-x README  0        0        2        1973
-rwxr-xr-x cat     0        0        3        14492
-rwxr-xr-x echo    0        0        4        13712
-rwxr-xr-x forktest 0        0        5        9428
-rwxr-xr-x grep    0        0        6        16136
-rwxr-xr-x init    0        0        7        14268
-rwxr-xr-x kill    0        0        8        13784
-rwxr-xr-x ln      0        0        9        13688
-rwxr-xr-x ls      0        0        10       17696
-rwxr-xr-x mkdir   0        0        11       13840
-rwxr-xr-x rm      0        0        12       13820
-rwxr-xr-x sh      0        0        13       27520
-rwxr-xr-x stressfs 0        0        14       14404
-rwxr-xr-x usertests 0        0        15       59592
-rwxr-xr-x wc      0        0        16       15004
-rwxr-xr-x zombie  0        0        17       13488
-rwxr-xr-x halt    0        0        18       13444
-rwxr-xr-x date    0        0        19       14976
-rwxr-xr-x ldtest  0        0        20       16948
-rwxr-xr-x ps      0        0        21       15424
-rwxr-xr-x time    0        0        22       14480
-rwxr-xr-x zombie_test 0        0        23       14168
-rwxr-xr-x loopforever 0        0        24       14132
-rwxr-xr-x ptest   0        0        25       14500
-rwxr-xr-x priotest 0        0        26       14360
-rwxr-xr-x chmod   0        0        27       14856
-rwxr-xr-x chown   0        0        28       14220
-rwxr-xr-x chgrp   0        0        29       14220
-rwxr-xr-x p5-test 0        0        30       28556
-rwxr-xr-x testsetuid 0        0        31       13608
crwxr-xr-x console 0        0        34        0
```

Figure 5: Invalid mode before

```
$ chmod 0857 date
ERROR: Mode is out of acceptable range 0000-1777.
$ ls
Mode      Name      UID      GID      iNode    Size
drwxr-xr-x .      0        0        1        1024
drwxr-xr-x ..     0        0        1        1024
-rwxr-xr-x README  0        0        2        1973
-rwxr-xr-x cat     0        0        3        14492
-rwxr-xr-x echo    0        0        4        13712
-rwxr-xr-x forktest 0        0        5        9428
-rwxr-xr-x grep    0        0        6        16136
-rwxr-xr-x init    0        0        7        14268
-rwxr-xr-x kill    0        0        8        13784
-rwxr-xr-x ln      0        0        9        13688
-rwxr-xr-x ls      0        0        10       17696
-rwxr-xr-x mkdir   0        0        11       13840
-rwxr-xr-x rm      0        0        12       13820
-rwxr-xr-x sh      0        0        13       27520
-rwxr-xr-x stressfs 0        0        14       14404
-rwxr-xr-x usertests 0        0        15       59592
-rwxr-xr-x wc      0        0        16       15004
-rwxr-xr-x zombie  0        0        17       13488
-rwxr-xr-x halt    0        0        18       13444
-rwxr-xr-x date    0        0        19       14976
-rwxr-xr-x ldtest  0        0        20       16948
-rwxr-xr-x ps      0        0        21       15424
-rwxr-xr-x time    0        0        22       14480
-rwxr-xr-x zombie_test 0        0        23       14168
-rwxr-xr-x loopforever 0        0        24       14132
-rwxr-xr-x ptest   0        0        25       14500
-rwxr-xr-x priotest 0        0        26       14360
-rwxr-xr-x chmod   0        0        27       14856
-rwxr-xr-x chown   0        0        28       14220
-rwxr-xr-x chgrp   0        0        29       14220
-rwxr-xr-x p5-test 0        0        30       28556
-rwxr-xr-x testsetuid 0        0        31       13608
crwxr-xr-x console 0        0        34        0
$
```

Figure 6: Invalid mode after

As we can see in Figure 5 and Figure 6 above, both before and after the chmod user command is given an invalid mode as input no changes have occurred. We can also see in Figure 6 that, upon being given an invalid mode as input, the chmod user command prints an appropriate error message stating that the mode given as input, 0857, is out of the appropriate range. The invalid mode sub-test PASSES. Because all chmod command sub-tests pass the chmod user command test PASSES as a whole.

2. chown user command test

In order to test that the chown user command functions properly testing will be split into three sub-tests:

- Valid parameters sub-test

In order to test that the chown user command successfully changes the UID of a file I will use the ls command to output the file information for files in xv6. I will then use the chown command and pass in a valid UID that is different than the default UID already displayed and a valid file name (time) currently in the ls output. I will then use the ls command again to see if the desired changes have occurred. I expect that the user command will successfully change the UID of time and that the changes will be noticeable in the ls output.

```
xv6...
cpu0: 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    1024
drwxr-xr-x ..         0         0         1    1024
-rwxr-xr-x README     0         0         2    1973
-rwxr-xr-x cat         0         0         3   14492
-rwxr-xr-x echo        0         0         4   13712
-rwxr-xr-x forktest    0         0         5    9428
-rwxr-xr-x grep        0         0         6   16136
-rwxr-xr-x init        0         0         7   14268
-rwxr-xr-x kill        0         0         8   13784
-rwxr-xr-x ln          0         0         9   13688
-rwxr-xr-x ls          0         0        10   17696
-rwxr-xr-x mkdir       0         0        11   13840
-rwxr-xr-x rm          0         0        12   13820
-rwxr-xr-x sh          0         0        13   27520
-rwxr-xr-x stressfs    0         0        14   14404
-rwxr-xr-x usertests   0         0        15   59592
-rwxr-xr-x wc          0         0        16   15004
-rwxr-xr-x zombie     0         0        17   13488
-rwxr-xr-x halt        0         0        18   13444
-rwxr-xr-x date        0         0        19   14976
-rwxr-xr-x idtest      0         0        20   16948
-rwxr-xr-x ps          0         0        21   15424
-rwxr-xr-x time        0         0        22   14480
-rwxr-xr-x zombieTest  0         0        23   14168
-rwxr-xr-x loopforever 0         0        24   14132
-rwxr-xr-x pstest      0         0        25   14500
-rwxr-xr-x priotest    0         0        26   14360
-rwxr-xr-x chmod       0         0        27   14856
-rwxr-xr-x chown       0         0        28   14220
-rwxr-xr-x chgrp       0         0        29   14220
-rwxr-xr-x ps-test     0         0        30   28556
-rwxr-xr-x testsetuid  0         0        31   13608
-rwxr-xr-x console     0         0        35     0
```

Figure 7: Valid parameters before

```
$ chown 4 time
$ ls
Mode                Name                UID      GID      INode  Size
drwxr-xr-x .          0         0         1    1024
drwxr-xr-x ..         0         0         1    1024
-rwxr-xr-x README     0         0         2    1973
-rwxr-xr-x cat         0         0         3   14492
-rwxr-xr-x echo        0         0         4   13712
-rwxr-xr-x forktest    0         0         5    9428
-rwxr-xr-x grep        0         0         6   16136
-rwxr-xr-x init        0         0         7   14268
-rwxr-xr-x kill        0         0         8   13784
-rwxr-xr-x ln          0         0         9   13688
-rwxr-xr-x ls          0         0        10   17696
-rwxr-xr-x mkdir       0         0        11   13840
-rwxr-xr-x rm          0         0        12   13820
-rwxr-xr-x sh          0         0        13   27520
-rwxr-xr-x stressfs    0         0        14   14404
-rwxr-xr-x usertests   0         0        15   59592
-rwxr-xr-x wc          0         0        16   15004
-rwxr-xr-x zombie     0         0        17   13488
-rwxr-xr-x halt        0         0        18   13444
-rwxr-xr-x date        0         0        19   14976
-rwxr-xr-x idtest      0         0        20   16948
-rwxr-xr-x ps          0         0        21   15424
-rwxr-xr-x time        4         0        22   14480
-rwxr-xr-x zombieTest  0         0        23   14168
-rwxr-xr-x loopforever 0         0        24   14132
-rwxr-xr-x pstest      0         0        25   14500
-rwxr-xr-x priotest    0         0        26   14360
-rwxr-xr-x chmod       0         0        27   14856
-rwxr-xr-x chown       0         0        28   14220
-rwxr-xr-x chgrp       0         0        29   14220
-rwxr-xr-x ps-test     0         0        30   28556
-rwxr-xr-x testsetuid  0         0        31   13608
-rwxr-xr-x console     0         0        35     0
$
```

Figure 8: Valid parameters after

As we can see in Figure 7 above, before the chown command is run, all processes have the same default UID value. After running the chown command and giving it two valid arguments: 4 as the UID and time for the file, the ls output verifies that the UID has been successfully changed for the time file and only the time file.

The valid parameters sub-test PASSES.

- Invalid filename sub-test

In order to test that the chown user command successfully handles an invalid file name I will use the ls command to output xv6 file information. I will then use the chown command and pass in a valid UID and an INVALID file name that is not currently in the ls output. I will then use the ls command again to see whether or not any changes have happened to existing files. I expect that the chown user command will successfully handle an invalid file name by printing an appropriate error message and that no changes will occur to any xv6 files which will be reinforced by no changes to the ls output.

```
xv6...
cpu0: 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     1024
drwxr-xr-x  ..          0        0        1     1024
-rwxr-xr-x  README      0        0        2     1973
-rwxr-xr-x  cat          0        0        3    14492
-rwxr-xr-x  echo         0        0        4    13712
-rwxr-xr-x  forktest    0        0        5     9428
-rwxr-xr-x  grep        0        0        6    16136
-rwxr-xr-x  init        0        0        7    14268
-rwxr-xr-x  kill        0        0        8    13784
-rwxr-xr-x  ln          0        0        9    13688
-rwxr-xr-x  ls          0        0       10    17696
-rwxr-xr-x  mkdir       0        0       11    13840
-rwxr-xr-x  rm          0        0       12    13820
-rwxr-xr-x  sh          0        0       13    27520
-rwxr-xr-x  stressfs    0        0       14    14404
-rwxr-xr-x  usertests   0        0       15    59592
-rwxr-xr-x  wc          0        0       16    15004
-rwxr-xr-x  zombie     0        0       17    13488
-rwxr-xr-x  halt       0        0       18    13444
-rwxr-xr-x  date       0        0       19    14976
-rwxr-xr-x  idtest     0        0       20    16948
-rwxr-xr-x  ps         0        0       21    15424
-rwxr-xr-x  time       0        0       22    14480
-rwxr-xr-x  zombieest  0        0       23    14168
-rwxr-xr-x  loopforever 0        0       24    14132
-rwxr-xr-x  pstest     0        0       25    14500
-rwxr-xr-x  priotest   0        0       26    14360
-rwxr-xr-x  chmod      0        0       27    14856
-rwxr-xr-x  chown      0        0       28    14220
-rwxr-xr-x  chgrp      0        0       29    14220
-rwxr-xr-x  ps-test    0        0       30    28556
-rwxr-xr-x  testsetuid  0        0       31    13608
-rwxr-xr-x  console    0        0       32        0
$
```

Figure 9: Invalid file name before

```
$ chown 4 hello
ERROR: chown failed.
$ ls
Mode                Name          UID      GID      iNode  Size
drwxr-xr-x  .           0        0        1     1024
drwxr-xr-x  ..          0        0        1     1024
-rwxr-xr-x  README      0        0        2     1973
-rwxr-xr-x  cat          0        0        3    14492
-rwxr-xr-x  echo         0        0        4    13712
-rwxr-xr-x  forktest    0        0        5     9428
-rwxr-xr-x  grep        0        0        6    16136
-rwxr-xr-x  init        0        0        7    14268
-rwxr-xr-x  kill        0        0        8    13784
-rwxr-xr-x  ln          0        0        9    13688
-rwxr-xr-x  ls          0        0       10    17696
-rwxr-xr-x  mkdir       0        0       11    13840
-rwxr-xr-x  rm          0        0       12    13820
-rwxr-xr-x  sh          0        0       13    27520
-rwxr-xr-x  stressfs    0        0       14    14404
-rwxr-xr-x  usertests   0        0       15    59592
-rwxr-xr-x  wc          0        0       16    15004
-rwxr-xr-x  zombie     0        0       17    13488
-rwxr-xr-x  halt       0        0       18    13444
-rwxr-xr-x  date       0        0       19    14976
-rwxr-xr-x  idtest     0        0       20    16948
-rwxr-xr-x  ps         0        0       21    15424
-rwxr-xr-x  time       0        0       22    14480
-rwxr-xr-x  zombieest  0        0       23    14168
-rwxr-xr-x  loopforever 0        0       24    14132
-rwxr-xr-x  pstest     0        0       25    14500
-rwxr-xr-x  priotest   0        0       26    14360
-rwxr-xr-x  chmod      0        0       27    14856
-rwxr-xr-x  chown      0        0       28    14220
-rwxr-xr-x  chgrp      0        0       29    14220
-rwxr-xr-x  ps-test    0        0       30    28556
-rwxr-xr-x  testsetuid  0        0       31    13608
-rwxr-xr-x  console    0        0       32        0
$
```

Figure 10: Invalid file name after

As we can see in Figure 9 above, before running the chown command with an invalid file name, all the files are set to their default UID values. After running the chown command and giving it two arguments: a valid UID and an INVALID file name, the chown command

successfully indicates failure which can be seen in Figure 10. The ls command also confirms that no information has been changed for any files. The invalid file name sub-test PASSES.

- Invalid UID sub-test

In order to test that the chown user command successfully handles an invalid UID I will use the ls command to output xv6 file information. I will then use the chown command and pass in an invalid UID and a valid file name that is currently in the ls output. I will then use the ls command again to see whether or not any changes have happened to existing files, specifically the one I targeted (time). I expect that the chown user command will successfully handle an invalid UID by printing an appropriate error message and that no changes will occur to any xv6 files which will be reinforced by no changes to the ls output.

```
xv6...
cpu0: starting
cpu1: starting
cpu2: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 ino
init: starting sh
$ ls
Mode          Name          UID          GID          iNode    Size
drwxr-xr-x .          0            0            1        1024
drwxr-xr-x ..         0            0            1        1024
-rwxr-xr-x README      0            0            2        1973
-rwxr-xr-x cat          0            0            3       14492
-rwxr-xr-x echo         0            0            4       13712
-rwxr-xr-x forktest     0            0            5        9428
-rwxr-xr-x grep         0            0            6       16136
-rwxr-xr-x init         0            0            7       14268
-rwxr-xr-x kill         0            0            8       13784
-rwxr-xr-x ln           0            0            9       13688
-rwxr-xr-x ls           0            0           10       17696
-rwxr-xr-x mkdir        0            0           11       13840
-rwxr-xr-x rm           0            0           12       13820
-rwxr-xr-x sh           0            0           13       27520
-rwxr-xr-x stressfs     0            0           14       14404
-rwxr-xr-x usertests    0            0           15       59592
-rwxr-xr-x wc           0            0           16       15804
-rwxr-xr-x zombie      0            0           17       13488
-rwxr-xr-x halt         0            0           18       13444
-rwxr-xr-x date         0            0           19       14976
-rwxr-xr-x ldttest      0            0           20       16948
-rwxr-xr-x ps           0            0           21       15424
-rwxr-xr-x time         0            0           22       14480
-rwxr-xr-x zombie-test  0            0           23       14168
-rwxr-xr-x loopforever  0            0           24       14132
-rwxr-xr-x ptest        0            0           25       14500
-rwxr-xr-x priotest     0            0           26       14360
-rwxr-xr-x chmod        0            0           27       14856
-rwxr-xr-x chown        0            0           28       14220
-rwxr-xr-x chgrp        0            0           29       14220
-rwxr-xr-x p5-test      0            0           30       28556
-rwxr-xr-x testsetuid   0            0           31       13608
-rwxr-xr-x console      0            0           32            0
$
```

Figure 11: Invalid UID before

```
$ chown -5 time
A valid UID is expected.
$ chown 38000 time
A valid UID is expected.
$ ls
Mode          Name          UID          GID          iNode    Size
drwxr-xr-x .          0            0            1        1024
drwxr-xr-x ..         0            0            1        1024
-rwxr-xr-x README      0            0            2        1973
-rwxr-xr-x cat          0            0            3       14492
-rwxr-xr-x echo         0            0            4       13712
-rwxr-xr-x forktest     0            0            5        9428
-rwxr-xr-x grep         0            0            6       16136
-rwxr-xr-x init         0            0            7       14268
-rwxr-xr-x kill         0            0            8       13784
-rwxr-xr-x ln           0            0            9       13688
-rwxr-xr-x ls           0            0           10       17696
-rwxr-xr-x mkdir        0            0           11       13840
-rwxr-xr-x rm           0            0           12       13820
-rwxr-xr-x sh           0            0           13       27520
-rwxr-xr-x stressfs     0            0           14       14404
-rwxr-xr-x usertests    0            0           15       59592
-rwxr-xr-x wc           0            0           16       15804
-rwxr-xr-x zombie      0            0           17       13488
-rwxr-xr-x halt         0            0           18       13444
-rwxr-xr-x date         0            0           19       14976
-rwxr-xr-x ldttest      0            0           20       16948
-rwxr-xr-x ps           0            0           21       15424
-rwxr-xr-x time         0            0           22       14480
-rwxr-xr-x zombie-test  0            0           23       14168
-rwxr-xr-x loopforever  0            0           24       14132
-rwxr-xr-x ptest        0            0           25       14500
-rwxr-xr-x priotest     0            0           26       14360
-rwxr-xr-x chmod        0            0           27       14856
-rwxr-xr-x chown        0            0           28       14220
-rwxr-xr-x chgrp        0            0           29       14220
-rwxr-xr-x p5-test      0            0           30       28556
-rwxr-xr-x testsetuid   0            0           31       13608
-rwxr-xr-x console      0            0           32            0
$
```

Figure 12: Invalid UID after

As we can see in Figure 11 above, before running the chown command with an invalid UID, all the files are set to their default UID values. After running the chown command and giving it two arguments: an invalid UID and an valid file name, the chown command successfully indicates failure which can be seen in Figure 12. This is shown for invalid UID's both below and above the acceptable range. The ls command also confirms that no information has been changed for any files including the time file I tested on. The invalid uid sub-test PASSES.

Because all sub-tests PASSED the chown user command test PASSES as a whole.

3. chgrp user command test

In order to test that the chgrp user command functions properly testing will be split into three sub-tests:

- Valid parameters sub-test

In order to test that the chgrp user command successfully changes the GID of a file I will use the ls command to output the file information for files in xv6. I will then use the chgrp command and pass in a valid GID that is different than the default GID already displayed and a valid file name (idtest) currently in the ls output. I will then use the ls command again to see if the desired changes have occurred. I expect that the user command will successfully change the GID of idtest and that the changes will be noticeable in the ls output.

```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inode
init: starting sh
$ ls
Mode                Name          UID      GID      iNode    Size
drwxr-xr-x .          0         0         1      1024
drwxr-xr-x ..         0         0         1      1024
-rwxr-xr-x README     0         0         2      1973
-rwxr-xr-x cat         0         0         3     14492
-rwxr-xr-x echo       0         0         4     13712
-rwxr-xr-x forktest   0         0         5      9428
-rwxr-xr-x grep       0         0         6     16136
-rwxr-xr-x init       0         0         7     14268
-rwxr-xr-x kill       0         0         8     13784
-rwxr-xr-x ln         0         0         9     13688
-rwxr-xr-x ls         0         0        10     17696
-rwxr-xr-x mkdir      0         0        11     13840
-rwxr-xr-x rm         0         0        12     13820
-rwxr-xr-x sh         0         0        13     27520
-rwxr-xr-x stressfs   0         0        14     14404
-rwxr-xr-x usertests  0         0        15     59592
-rwxr-xr-x wc         0         0        16     15004
-rwxr-xr-x zombie     0         0        17     13488
-rwxr-xr-x halt       0         0        18     13444
-rwxr-xr-x date       0         0        19     14976
-rwxr-xr-x idtest     0         0        20     16948
-rwxr-xr-x ps         0         0        21     15424
-rwxr-xr-x time       0         0        22     14480
-rwxr-xr-x zombieTest 0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest      0         0        25     14500
-rwxr-xr-x priotest   0         0        26     14360
-rwxr-xr-x chmod      0         0        27     14856
-rwxr-xr-x chown      0         0        28     14220
-rwxr-xr-x chgrp      0         0        29     14220
-rwxr-xr-x ps-test    0         0        30     28556
-rwxr-xr-x testsetuid 0         0        31     13608
crwxr-xr-x console   0         0        32         0
$
```

Figure 13: Valid parameters before

```
$ chgrp 4 idtest
$ ls
Mode                Name          UID      GID      iNode    Size
drwxr-xr-x .          0         0         1      1024
drwxr-xr-x ..         0         0         1      1024
-rwxr-xr-x README     0         0         2      1973
-rwxr-xr-x cat         0         0         3     14492
-rwxr-xr-x echo       0         0         4     13712
-rwxr-xr-x forktest   0         0         5      9428
-rwxr-xr-x grep       0         0         6     16136
-rwxr-xr-x init       0         0         7     14268
-rwxr-xr-x kill       0         0         8     13784
-rwxr-xr-x ln         0         0         9     13688
-rwxr-xr-x ls         0         0        10     17696
-rwxr-xr-x mkdir      0         0        11     13840
-rwxr-xr-x rm         0         0        12     13820
-rwxr-xr-x sh         0         0        13     27520
-rwxr-xr-x stressfs   0         0        14     14404
-rwxr-xr-x usertests  0         0        15     59592
-rwxr-xr-x wc         0         0        16     15004
-rwxr-xr-x zombie     0         0        17     13488
-rwxr-xr-x halt       0         0        18     13444
-rwxr-xr-x date       0         0        19     14976
-rwxr-xr-x idtest     0         4        20     16948
-rwxr-xr-x ps         0         0        21     15424
-rwxr-xr-x time       0         0        22     14480
-rwxr-xr-x zombieTest 0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest      0         0        25     14500
-rwxr-xr-x priotest   0         0        26     14360
-rwxr-xr-x chmod      0         0        27     14856
-rwxr-xr-x chown      0         0        28     14220
-rwxr-xr-x chgrp      0         0        29     14220
-rwxr-xr-x ps-test    0         0        30     28556
-rwxr-xr-x testsetuid 0         0        31     13608
crwxr-xr-x console   0         0        32         0
$
```

Figure 14: Valid parameters after

As we can see in Figure 13 above, before the chgrp command is run, all processes have the same default GID value. After running the chgrp command and giving it two valid arguments: 4 as the GID and idtest for the file, the ls output verifies that the GID has been successfully changed for the idtest file and only the idtest file.

The valid parameters sub-test PASSES.

- Invalid file name sub-test

In order to test that the chgrp user command successfully handles an invalid file name I will use the ls command to output xv6 file information. I will then use the chgrp command and pass in a valid GID and an INVALID file name that is not currently in the ls output. I will then use the ls command again to see whether or not any changes have happened to existing files. I expect that the chgrp user command will successfully handle an invalid file name by printing an appropriate error message and that no changes will occur to any xv6 files which will be reinforced by no changes to the ls output.

```
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inode
init: starting sh
$ ls
Mode          Name          UID      GID      iNode    Size
drwxr-xr-x .          0         0         1      1024
drwxr-xr-x ..         0         0         1      1024
-rwxr-xr-x README     0         0         2       1973
-rwxr-xr-x cat         0         0         3     14492
-rwxr-xr-x echo        0         0         4     13712
-rwxr-xr-x forktest    0         0         5      9428
-rwxr-xr-x grep        0         0         6     16136
-rwxr-xr-x init        0         0         7     14268
-rwxr-xr-x kill        0         0         8     13784
-rwxr-xr-x ln          0         0         9     13688
-rwxr-xr-x ls          0         0        10     17696
-rwxr-xr-x mkdir       0         0        11     13840
-rwxr-xr-x rm          0         0        12     13820
-rwxr-xr-x sh          0         0        13     27520
-rwxr-xr-x stressfs    0         0        14     14404
-rwxr-xr-x usertests   0         0        15     59592
-rwxr-xr-x wc          0         0        16     15004
-rwxr-xr-x zombie      0         0        17     13488
-rwxr-xr-x halt        0         0        18     13444
-rwxr-xr-x date        0         0        19     14976
-rwxr-xr-x ldttest     0         0        20     16948
-rwxr-xr-x ps          0         0        21     15424
-rwxr-xr-x time        0         0        22     14480
-rwxr-xr-x zombie      0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest       0         0        25     14500
-rwxr-xr-x priotest    0         0        26     14360
-rwxr-xr-x chmod       0         0        27     14856
-rwxr-xr-x chown       0         0        28     14220
-rwxr-xr-x chgrp       0         0        29     14220
-rwxr-xr-x p5-test     0         0        30     28556
-rwxr-xr-x testsetuid  0         0        31     13608
-rwxr-xr-x console     0         0        32         0
```

Figure 15: Invalid file name before

```
$ chgrp 4 helloworld
ERROR: chgrp failed.
$ ls
Mode          Name          UID      GID      iNode    Size
drwxr-xr-x .          0         0         1      1024
drwxr-xr-x ..         0         0         1      1024
-rwxr-xr-x README     0         0         2       1973
-rwxr-xr-x cat         0         0         3     14492
-rwxr-xr-x echo        0         0         4     13712
-rwxr-xr-x forktest    0         0         5      9428
-rwxr-xr-x grep        0         0         6     16136
-rwxr-xr-x init        0         0         7     14268
-rwxr-xr-x kill        0         0         8     13784
-rwxr-xr-x ln          0         0         9     13688
-rwxr-xr-x ls          0         0        10     17696
-rwxr-xr-x mkdir       0         0        11     13840
-rwxr-xr-x rm          0         0        12     13820
-rwxr-xr-x sh          0         0        13     27520
-rwxr-xr-x stressfs    0         0        14     14404
-rwxr-xr-x usertests   0         0        15     59592
-rwxr-xr-x wc          0         0        16     15004
-rwxr-xr-x zombie      0         0        17     13488
-rwxr-xr-x halt        0         0        18     13444
-rwxr-xr-x date        0         0        19     14976
-rwxr-xr-x ldttest     0         0        20     16948
-rwxr-xr-x ps          0         0        21     15424
-rwxr-xr-x time        0         0        22     14480
-rwxr-xr-x zombie      0         0        23     14168
-rwxr-xr-x loopforever 0         0        24     14132
-rwxr-xr-x ptest       0         0        25     14500
-rwxr-xr-x priotest    0         0        26     14360
-rwxr-xr-x chmod       0         0        27     14856
-rwxr-xr-x chown       0         0        28     14220
-rwxr-xr-x chgrp       0         0        29     14220
-rwxr-xr-x p5-test     0         0        30     28556
-rwxr-xr-x testsetuid  0         0        31     13608
-rwxr-xr-x console     0         0        32         0
```

Figure 16: Invalid file name after

As we can see in Figure 15 above, before running the chgrp command with an invalid file name, all the files are set to their default GID values. After running the chgrp command and giving it two arguments: a valid GID and an INVALID file name, the chgrp command successfully indicates failure which can be seen in Figure 16. The ls command also confirms that no information has been changed for any files.

The invalid file name sub-test PASSES.

- Invalid GID sub-test

In order to test that the chgrp user command successfully handles an invalid GID I will use the ls command to output xv6 file information. I will then use the chgrp command and pass in an invalid GID and a valid file name that is currently in the ls output. I will then use the ls command again to see whether or not any changes have happened to existing files, specifically the one I targeted (idtest). I expect that the chgrp user command will successfully handle an invalid GID by printing an appropriate error message and that no changes will occur to any xv6 files which will be reinforced by no changes in the ls output.

```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inode
init: starting sh
$ ls
Mode          Name          UID      GID      iNode  Size
drwxr-xr-x .          0        0        1      1024
drwxr-xr-x ..         0        0        1      1024
-rwxr-xr-x README      0        0        2      1973
-rwxr-xr-x cat          0        0        3      14492
-rwxr-xr-x echo         0        0        4      13712
-rwxr-xr-x forktest     0        0        5      9428
-rwxr-xr-x grep         0        0        6      16136
-rwxr-xr-x init         0        0        7      14268
-rwxr-xr-x kill         0        0        8      13784
-rwxr-xr-x ln           0        0        9      13688
-rwxr-xr-x ls           0        0       10     17696
-rwxr-xr-x mkdir        0        0       11     13840
-rwxr-xr-x rm           0        0       12     13820
-rwxr-xr-x sh           0        0       13     27520
-rwxr-xr-x stressfs     0        0       14     14404
-rwxr-xr-x usertests    0        0       15     59592
-rwxr-xr-x wc           0        0       16     15004
-rwxr-xr-x zombie       0        0       17     13488
-rwxr-xr-x halt         0        0       18     13444
-rwxr-xr-x date         0        0       19     14976
-rwxr-xr-x idtest       0        0       20     16948
-rwxr-xr-x ps           0        0       21     15424
-rwxr-xr-x time         0        0       22     14480
-rwxr-xr-x zombieTest   0        0       23     14168
-rwxr-xr-x loopforever  0        0       24     14132
-rwxr-xr-x ptest        0        0       25     14500
-rwxr-xr-x priotest     0        0       26     14360
-rwxr-xr-x chmod        0        0       27     14856
-rwxr-xr-x chown        0        0       28     14220
-rwxr-xr-x chgrp        0        0       29     14220
-rwxr-xr-x ps-test      0        0       30     28556
-rwxr-xr-x testsetuid   0        0       31     13608
crwxr-xr-x console     0        0       32        0
```

Figure 17: Invalid GID before

```
$ chgrp -1 idtest
A valid GID is expected.
$ chgrp 38000 idtest
A valid GID is expected.
$ ls
Mode          Name          UID      GID      iNode  Size
drwxr-xr-x .          0        0        1      1024
drwxr-xr-x ..         0        0        1      1024
-rwxr-xr-x README      0        0        2      1973
-rwxr-xr-x cat          0        0        3      14492
-rwxr-xr-x echo         0        0        4      13712
-rwxr-xr-x forktest     0        0        5      9428
-rwxr-xr-x grep         0        0        6      16136
-rwxr-xr-x init         0        0        7      14268
-rwxr-xr-x kill         0        0        8      13784
-rwxr-xr-x ln           0        0        9      13688
-rwxr-xr-x ls           0        0       10     17696
-rwxr-xr-x mkdir        0        0       11     13840
-rwxr-xr-x rm           0        0       12     13820
-rwxr-xr-x sh           0        0       13     27520
-rwxr-xr-x stressfs     0        0       14     14404
-rwxr-xr-x usertests    0        0       15     59592
-rwxr-xr-x wc           0        0       16     15004
-rwxr-xr-x zombie       0        0       17     13488
-rwxr-xr-x halt         0        0       18     13444
-rwxr-xr-x date         0        0       19     14976
-rwxr-xr-x idtest       0        0       20     16948
-rwxr-xr-x ps           0        0       21     15424
-rwxr-xr-x time         0        0       22     14480
-rwxr-xr-x zombieTest   0        0       23     14168
-rwxr-xr-x loopforever  0        0       24     14132
-rwxr-xr-x ptest        0        0       25     14500
-rwxr-xr-x priotest     0        0       26     14360
-rwxr-xr-x chmod        0        0       27     14856
-rwxr-xr-x chown        0        0       28     14220
-rwxr-xr-x chgrp        0        0       29     14220
-rwxr-xr-x ps-test      0        0       30     28556
-rwxr-xr-x testsetuid   0        0       31     13608
crwxr-xr-x console     0        0       32        0
$
```

Figure 18: Invalid GID after

As we can see in Figure 17 above, before running the chgrp command with an invalid GID, all the files are set to their default GID values. After running the chgrp command and giving it two arguments: an invalid GID and a valid file name, the chgrp command successfully indicates failure which can be seen in Figure 18. This is shown for invalid GID's both below and above the acceptable range. The ls command also confirms that no information has been changed for any files including the idtest file I tested on. The invalid gid sub-test PASSES.

Because all sub-tests PASSED the chgrp user command test PASSES as a whole.

4. ls command test

In order to test that the ls command is outputting correct information I will do an initial run of the ls command with the default uid, gid, and mode values in param.h. I will then change the value of the default uid, gid, and mode values to different values, do a make clean, and then boot xv6 and use the ls command again to see if any information has changed. I will also test ls given a single argument. I expect that, after changing the default values for uid, gid, and mode, the ls command will output that new information for each file in its output.

```
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 1024
drwxr-xr-x .. 0 0 1 1024
-rwxr-xr-x README 0 0 2 1973
-rwxr-xr-x cat 0 0 3 14492
-rwxr-xr-x echo 0 0 4 13712
-rwxr-xr-x forktest 0 0 5 9428
-rwxr-xr-x grep 0 0 6 16136
-rwxr-xr-x init 0 0 7 14268
-rwxr-xr-x kill 0 0 8 13784
-rwxr-xr-x ln 0 0 9 13688
-rwxr-xr-x ls 0 0 10 17696
-rwxr-xr-x mkdir 0 0 11 13840
-rwxr-xr-x rm 0 0 12 13820
-rwxr-xr-x sh 0 0 13 27520
-rwxr-xr-x stressfs 0 0 14 14404
-rwxr-xr-x usertests 0 0 15 59592
-rwxr-xr-x wc 0 0 16 15004
-rwxr-xr-x zombie 0 0 17 13488
-rwxr-xr-x halt 0 0 18 13444
-rwxr-xr-x date 0 0 19 14976
-rwxr-xr-x idtest 0 0 20 16948
-rwxr-xr-x ps 0 0 21 15424
-rwxr-xr-x time 0 0 22 14480
-rwxr-xr-x zombie 0 0 23 14168
-rwxr-xr-x loopforever 0 0 24 14132
-rwxr-xr-x ptest 0 0 25 14500
-rwxr-xr-x priotest 0 0 26 14360
-rwxr-xr-x chmod 0 0 27 14856
-rwxr-xr-x chown 0 0 28 14220
-rwxr-xr-x chgrp 0 0 29 14220
-rwxr-xr-x ps-test 0 0 30 28556
-rwxr-xr-x testsetuid 0 0 31 13608
-rwxr-xr-x console 0 0 32 0
$
```

Figure 19: ls test before default change

```
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
d--wx-wx-wx . 20 20 1 1024
d--wx-wx-wx .. 20 20 1 1024
--wx-wx-wx README 20 20 2 1973
--wx-wx-wx cat 20 20 3 14492
--wx-wx-wx echo 20 20 4 13712
--wx-wx-wx forktest 20 20 5 9428
--wx-wx-wx grep 20 20 6 16136
--wx-wx-wx init 20 20 7 14268
--wx-wx-wx kill 20 20 8 13784
--wx-wx-wx ln 20 20 9 13688
--wx-wx-wx ls 20 20 10 17696
--wx-wx-wx mkdir 20 20 11 13840
--wx-wx-wx rm 20 20 12 13820
--wx-wx-wx sh 20 20 13 27520
--wx-wx-wx stressfs 20 20 14 14404
--wx-wx-wx usertests 20 20 15 59592
--wx-wx-wx wc 20 20 16 15004
--wx-wx-wx zombie 20 20 17 13488
--wx-wx-wx halt 20 20 18 13444
--wx-wx-wx date 20 20 19 14976
--wx-wx-wx idtest 20 20 20 16948
--wx-wx-wx ps 20 20 21 15424
--wx-wx-wx time 20 20 22 14480
--wx-wx-wx zombie 20 20 23 14168
--wx-wx-wx loopforever 20 20 24 14132
--wx-wx-wx ptest 20 20 25 14500
--wx-wx-wx priotest 20 20 26 14360
--wx-wx-wx chmod 20 20 27 14856
--wx-wx-wx chown 20 20 28 14220
--wx-wx-wx chgrp 20 20 29 14220
--wx-wx-wx ps-test 20 20 30 28556
--wx-wx-wx testsetuid 20 20 31 13608
c--wx-wx-wx console 20 20 32 0
$
```

Figure 20: ls test after default change

As we can see in Figure 19 above, the original default value for the uid, gid, and mode are set to 0, 0, and 00755 respectively. I chose to change the default value for the uid, gid, and mode to 20, 20, and 00333 respectively. After the change was made and a make clean was performed the changes to the defaults can be seen in Figure 20 above. The ls command output has changed to accurately reflect the changes in the mode column, the uid column, and the gid column for all of the files listed. We can also see that ls works when given a single file argument (other figure). The ls test PASSES.

```
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 1024
drwxr-xr-x .. 0 0 1 1024
-rwxr-xr-x README 0 0 2 1973
-rwxr-xr-x cat 0 0 3 14492
-rwxr-xr-x echo 0 0 4 13712
-rwxr-xr-x forktest 0 0 5 9428
-rwxr-xr-x grep 0 0 6 16136
-rwxr-xr-x init 0 0 7 14268
-rwxr-xr-x kill 0 0 8 13784
-rwxr-xr-x ln 0 0 9 13688
-rwxr-xr-x ls 0 0 10 17696
-rwxr-xr-x mkdir 0 0 11 13840
-rwxr-xr-x rm 0 0 12 13820
-rwxr-xr-x sh 0 0 13 27520
-rwxr-xr-x stressfs 0 0 14 14404
-rwxr-xr-x usertests 0 0 15 59592
-rwxr-xr-x wc 0 0 16 15004
-rwxr-xr-x zombie 0 0 17 13488
-rwxr-xr-x halt 0 0 18 13444
-rwxr-xr-x date 0 0 19 14976
-rwxr-xr-x idtest 0 0 20 16948
-rwxr-xr-x ps 0 0 21 15424
-rwxr-xr-x time 0 0 22 14480
-rwxr-xr-x zombie 0 0 23 14168
-rwxr-xr-x loopforever 0 0 24 14132
-rwxr-xr-x ptest 0 0 25 14500
-rwxr-xr-x priotest 0 0 26 14360
-rwxr-xr-x chmod 0 0 27 14856
-rwxr-xr-x chown 0 0 28 14220
-rwxr-xr-x chgrp 0 0 29 14220
-rwxr-xr-x ps-test 0 0 30 28556
-rwxr-xr-x testsetuid 0 0 31 13608
-rwxr-xr-x console 0 0 32 0
$ ls halt
Mode          Name          UID      GID      iNode  Size
-rwxr-xr-x halt 0 0 18 13444
$
```

Other figure: ls given file argument

5. persistence testing

In order to test whether or not persistence is occurring in the xv6 operating system I will boot xv6 and use the `ls` command to output the default information for each of the files. I will then make modifications to a couple of the files including modifications to their uid, gid, and mode values. I will use the `ls` command again to output the changes. I will then use the `halt` command to shut down xv6 and will then boot xv6 again. I will then use the `ls` command and check to see whether or not the initial changes that I made are still there. I expect that, after making changes to a few of the files and rebooting xv6, the same changes will have persisted past shutting xv6 down and rebooting.

```
xv6...
cpu0: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inode
init: starting sh
$ ls
Mode          Name          UID      GID      iNode    Size
drwxr-xr-x .          0        0        1      1024
drwxr-xr-x ..         0        0        1      1024
-rwxr-xr-x README     0        0        2      1973
-rwxr-xr-x cat         0        0        3      14492
-rwxr-xr-x echo        0        0        4      13712
-rwxr-xr-x forktest    0        0        5      9428
-rwxr-xr-x grep        0        0        6      16136
-rwxr-xr-x init        0        0        7      14268
-rwxr-xr-x kill        0        0        8      13784
-rwxr-xr-x ln          0        0        9      13688
-rwxr-xr-x ls          0        0       10     17696
-rwxr-xr-x mkdir       0        0       11     13840
-rwxr-xr-x rm          0        0       12     13820
-rwxr-xr-x sh          0        0       13     27520
-rwxr-xr-x stressfs    0        0       14     14404
-rwxr-xr-x usertests   0        0       15     59592
-rwxr-xr-x wc          0        0       16     15004
-rwxr-xr-x zombie     0        0       17     13488
-rwxr-xr-x halt        0        0       18     13444
-rwxr-xr-x date        0        0       19     14976
-rwxr-xr-x ldttest     0        0       20     16948
-rwxr-xr-x ps          0        0       21     15424
-rwxr-xr-x time        0        0       22     14480
-rwxr-xr-x zombie      0        0       23     14168
-rwxr-xr-x loopforever 0        0       24     14132
-rwxr-xr-x ptest       0        0       25     14500
-rwxr-xr-x priotest    0        0       26     14360
-rwxr-xr-x chmod       0        0       27     14856
-rwxr-xr-x chown       0        0       28     14220
-rwxr-xr-x chgrp       0        0       29     14220
-rwxr-xr-x ps-test     0        0       30     28556
-rwxr-xr-x testsetuid  0        0       31     13608
-rwxr-xr-x console     0        0       32        0
$
```

Figure 21: Persistence before

```
$ chmod 0333 date
String in octal is 0333 and decimal is 219
$ chown 4 date
main-loop: WARNING: I/O thread spun for 1000 iterations
$ chgrp 4 date
$ chmod 0357 time
String in octal is 0357 and decimal is 239
$ chown 4 time
$ chgrp 4 time
$ ls
Mode          Name          UID      GID      iNode    Size
drwxr-xr-x .          0        0        1      1024
drwxr-xr-x ..         0        0        1      1024
-rwxr-xr-x README     0        0        2      1973
-rwxr-xr-x cat         0        0        3      14492
-rwxr-xr-x echo        0        0        4      13712
-rwxr-xr-x forktest    0        0        5      9428
-rwxr-xr-x grep        0        0        6      16136
-rwxr-xr-x init        0        0        7      14268
-rwxr-xr-x kill        0        0        8      13784
-rwxr-xr-x ln          0        0        9      13688
-rwxr-xr-x ls          0        0       10     17696
-rwxr-xr-x mkdir       0        0       11     13840
-rwxr-xr-x rm          0        0       12     13820
-rwxr-xr-x sh          0        0       13     27520
-rwxr-xr-x stressfs    0        0       14     14404
-rwxr-xr-x usertests   0        0       15     59592
-rwxr-xr-x wc          0        0       16     15004
-rwxr-xr-x zombie     0        0       17     13488
-rwxr-xr-x halt        0        0       18     13444
-rwxr-xr-x date        4        4       19     14976
-rwxr-xr-x ldttest     0        0       20     16948
-rwxr-xr-x ps          0        0       21     15424
-rwxr-xr-x time        4        4       22     14480
-rwxr-xr-x zombie      0        0       23     14168
-rwxr-xr-x loopforever 0        0       24     14132
-rwxr-xr-x ptest       0        0       25     14500
-rwxr-xr-x priotest    0        0       26     14360
-rwxr-xr-x chmod       0        0       27     14856
-rwxr-xr-x chown       0        0       28     14220
-rwxr-xr-x chgrp       0        0       29     14220
-rwxr-xr-x ps-test     0        0       30     28556
-rwxr-xr-x testsetuid  0        0       31     13608
-rwxr-xr-x console     0        0       32        0
$
```

Figure 22: Persistence after changes

```

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    1024
drwxr-xr-x ..         0         0         1    1024
-rwxr-xr-x README     0         0         2    1973
-rwxr-xr-x cat        0         0         3   14492
-rwxr-xr-x echo       0         0         4   13712
-rwxr-xr-x forktest   0         0         5    9428
-rwxr-xr-x grep       0         0         6   16136
-rwxr-xr-x init       0         0         7   14268
-rwxr-xr-x kill       0         0         8   13784
-rwxr-xr-x ln         0         0         9   13688
-rwxr-xr-x ls         0         0        10   17696
-rwxr-xr-x mkdir     0         0        11   13840
-rwxr-xr-x rm         0         0        12   13820
-rwxr-xr-x sh         0         0        13   27520
-rwxr-xr-x stressfs   0         0        14   14404
-rwxr-xr-x usertests   0         0        15  59592
-rwxr-xr-x wc         0         0        16   15004
-rwxr-xr-x zombie     0         0        17   13488
-rwxr-xr-x halt       0         0        18   13444
--wx-wx-wx date       4         4        19   14976
-rwxr-xr-x idtest     0         0        20   16948
-rwxr-xr-x ps         0         0        21   15424
--wxr-xrwx time       4         4        22  14480
-rwxr-xr-x zombieTest 0         0        23   14168
-rwxr-xr-x loopforever 0         0        24   14132
-rwxr-xr-x pstest     0         0        25   14500
-rwxr-xr-x priotest   0         0        26   14360
-rwxr-xr-x chmod      0         0        27   14856
-rwxr-xr-x chown      0         0        28   14220
-rwxr-xr-x chgrp      0         0        29   14220
-rwxr-xr-x p5-test    0         0        30  28556
-rwxr-xr-x testsetuid 0         0        31   13608
crwxr-xr-x console    0         0        33     0
$

```

Figure 23: Persistence after reboot

As we can see in Figure 21 above, before any changes have been made to any of the xv6 files, they all have the default mode, uid, and gid values. After changing the mode, uid, and gid of date to 0333, 4, and 4 respectively and changing the mode, uid, and gid of time to 0357, 4, and 4 respectively we can see in Figure 22 that the changes were made successfully when the ls command outputs file information. We can see in Figure 23 that, after using the halt command and rebooting xv6, then using the ls command, the changes made to the date and time files have persisted even after the reboot.

The persistence test PASSES.

P5-TEST

Additional testing for the newly added/modified system calls will be done using the p5-test suite provided for us in this project. These can be found on the next page.

1. chmod() system call test

In order to test that the chmod() system call is functioning properly I will use the chmod() option (#3) in the p5-test suite.

The way the chmod test tests the chmod system call is by calling the doChmodTest() function, passing in testsetuid as the file that it calls the chmod system call on. First, doChmodTest() grabs the inode information for the testsetuid file and stores it in a stat variable named st. A mode variable is then set to the initial mode value for the testsetuid file which should be the default value of 0755. A loop is then set up and loops NUMPERMSTOCHECK times. Each time the loop happens a few things occur:

- The `chmod()` system call is called, passing in `testsetuid` as its pathname and passing in an argument from the `perms[]` array declared in `p5-test.h` as the mode value that it attempts to change `testsetuid`'s mode value to.
- After the `chmod()` system call is called, the `stat` function is run to re-acquire the inode information for the `testsetuid` file. The mode value after the `chmod` system call is stored in a variable called `testmode`. The reason for doing this is because, if `chmod` was successful in changing the mode for the `testsetuid` file it should now be different than the original mode that we set the mode variable to in the beginning of `doChmodTest()`.
- A check is then performed to compare the original mode that `testsetuid` had to the mode that `testsetuid` has after the `chmod` system call has been called on `testsetuid` (`testmode`). If the original mode is the same as `testmode` this means that the `chmod()` system call was NOT successful in changing the mode, an error message is printed to indicate this, and `doChmodTest()` is exited.
- If the mode has changed successfully the loop continues and runs again, repeating the above steps but with a different mode value used from the `perms[]` array in `p5-test.h`
- Once the loop has finished the mode is changed back to the default value of `0755`, a success message is printed, and a value of `1 (PASS)` is returned to indicate success.

This means that, if the `chmod()` test successfully completes execution and indicates success, that the `chmod()` system call successfully changes the mode when given valid file and mode inputs.

```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 194
init: starting sh
$ p5-test

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

Executing chmod() test.

Test Passed

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

Figure 24: chmod system call

As we can see in Figure 24 above, the `chmod()` test successfully passed without indicating failure.

The `chmod()` system call test PASSES.

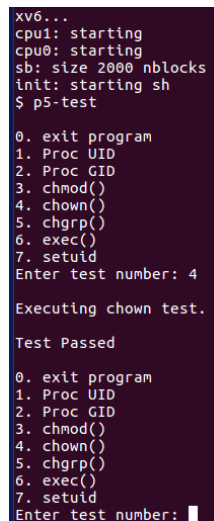
2. Chown() system call test

In order to test that the chown() system call is functioning properly I will use the chown() option (#4) in the p5-test suite.

The way the chown test works is by calling the doChownTest() function, passing in testsetuid as the file that it calls the chown() system call on.

- First, the stat function is called to retrieve testsetuid's inode information and stores it in the st variable.
- Then, uid1 is set to the initial uid value for testsetuid's inode information.
- The chown system call is run passing in testsetuid as the file and passing in uid1+1 as the uid value to change testsetuid's uid to. In essence, it changes the uid value to the uid one greater than it.
- The return value of the chown system call is checked and, if chown indicates failure for any reason, an error message is printed indicating that chown failed to set the uid and NOPASS (a value of 0) is returned to indicate doChownTest() failed.
- The stat function is run to re-acquire testsetuid's inode information. The uid2 variable is set to the value of testsetuid's uid after the chown system call has been called.
- Uid1 is compared to uid2 to see if the two values are the same or different. If the two values are the same this means that the chown system call failed to change the uid for testsetuid. An error message is printed stating that the values should differ after the chown system call is run, and NOPASS (a value of 0) is returned to indicate doChownTest() failed.
- If the two values are different this means that the chown system call successfully changed the uid of the testsetuid file. The chown system call is then run to change the uid value for testsetuid back to the original value, a success message is printed, and a PASS (value of 1) is returned to indicate doChownTest() was successful.

This means that, if the chown() test in the p5-test suite runs and indicates success, that the chown system call successfully passes its test.



```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks
init: starting sh
$ p5-test

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

Executing chown test.

Test Passed

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

Figure 25: chown system call test

As we can see in Figure 25 above, the chown() test in the p5-test suite ran successfully without indicating failure. This means that the chown system call successfully changes the uid of the file passed in to the uid argument passed in.

The chown system call test PASSES.

3. chgrp() test

In order to test that the chgrp() system call is functioning properly I will use the chgrp() option (#5) in the p5-test suite.

The chgrp() test in the p5-test suite works using the exact same logic as the chown() test described above, except that it sets gid1 to the initial value of testsetuid's gid, runs the chgrp() system call using testsetuid as the file and gid1+1 as the value to change testsetuid's gid to. It then compares gid1 (the initial gid value) to gid2 (the value after the chgrp() system call is called). If the two values are the same this means that the chgrp() system call failed to change testsetuid's gid value to the new value. An error message is printed and NOPASS (a value of 0) is returned. If the gid1 and gid2 values differ, this means that the chgrp() system call successfully changed testsetuid's gid value to the value passed into the chgrp() system call. A success message is printed and PASS (a value of 1) is returned to indicate doChgrpTest was successful.

This means that, if the chgrp() test in the p5-test suite runs and indicates success, that the chgrp system call successfully passes its test.

```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks
init: starting sh
$ p5-test

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

Executing chgrp test.

Test Passed

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

Figure 26: chgrp system call test

As we can see in Figure 26 above, the chgrp() test in the p5-test suite ran successfully without indicating failure. This means that the chgrp system call successfully changes the gid of the file passed in to the gid argument passed in.

The chgrp system call test PASSES.

4. **exec() system call test**

In order to test that the `exec()` system call is using protection properly I will use the `exec()` test provided in the p5-test suite.

The `exec()` test in the p5-test suite works by calling the `doExecTest()` function passing in `testsetuid` as the file that it operates on. First, `doExecTest()` calls the `canRun()` function, passing in `testsetuid` as its argument, in order to make sure that `testsetuid` can even be executed before the overall test begins. The default mode value 0755 ensures that it can be run. The `stat` function is used to retrieve `testsetuid`'s initial inode information. `UID` and `GID` variables are set to the initial `uid` and `gid` of the `testsetuid` file to preserve them. A loop, looping `NUMPERMSTOCHECK` times, is then run to do the bulk of the testing:

- An array matrix called `testperms` provided in `p5-test.h` provides `uid` and `gid` values for the currently running process and the `testsetuid` file in order to ensure that the `exec` system call hits all cases in which it should check the execute bits of `testsetuid`'s mode value for execute permission (user then group then other). The `perms[]` array that contains the mode values used for execute permissions contains a final mode value with no execute permissions.
- The `setuid()` and `setgid()` system calls are used to set the `uid` and `gid` of the currently running process to values in the `testperms` matrix corresponding to the current for loop iteration.
- The `chown()`, `chgrp()`, and `chmod()` system calls are then used to set the `testsetuid` file's `uid` and `gid` to the values provided in this matrix as well as set its mode to a value in the `perms` array corresponding to the current for loop iteration.
- A child process is then forked. If the fork is successful the child process runs the `exec` system call with the `uid`, `gid`, and mode that were set in the steps prior to this. Since each `uid`, `gid`, and mode in the `testperms` matrix and `perms` array provide cases where the file has execute permission either in the user, group, or other execute bit EXCEPT for the last mode in the `perms` array... if the `exec` system call indicates failure on any iteration other than the last iteration it means that the `exec` system call failed when it shouldn't have and an error message is printed before the child process exits back to the parent process. The only time `exec` should indicate failure is on the last mode value in the `perms` array which means that the `exec` command SUCCESSFULLY indicated failure. The error message printed in that case reiterates this concept by printing that `exec` failed AS EXPECTED.
- The parent process waits for its child to finish executing before continuing on to the next iteration through the loop.

After the final iteration through the for loop is completed the `chown()`, `chgrp()`, and `chmod()` system calls are used to set `testsetuid`'s `uid`, `gid`, and mode back to the default values and `PASS` (return value of 1) is return.

All the logic above combined means that, if the `exec()` test visually indicates failure for any of the output OTHER THAN the last `exec` call which should fail as expected, then the `exec()` test failed. However, I expect that there will be no failure other than in the last case that is expected to fail.

```

xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logsta
init: starting sh
$ p5-test

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

Executing exec test.

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

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

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

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

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

```

Figure 27: exec system call test

As we can see in Figure 27 above, upon visual inspection of the output from the `exec()` test in the p5-test suite, the `exec()` system call only indicated failure in the case that it was expected to do so. All other tests completed successfully.

The `exec` system call test PASSES.

5. **setuid test**

In order to test that `setuid` bit is being set properly I will use the `setuid` test in the p5-test suite.

The way that this test works is exactly the same way that the `exec()` test in the p5-test suite works which is precisely described above; short of one minor difference. In the `setuid` test which calls `doSetuidTest()`, the `uid` and `gid` information are printed out for both the currently running process and the file it is trying to run; in this case `testsetuid`. The mode value is also printed out. This allows us to see the `uid` and `gid` of both the process and the file before the execution of the file. It allows us to see that, when the `testsetuid` file is executed, which simply prints the process' `uid`, the `uid` has changed to take on the `uid` of the file when the `setuid` bit for the file's mode value is a 1. However, this only occurs when the process has execute permission as well.

In the case of this test, all the test modes provided in the `perms` array have a value of 1 for their `setuid` bit so, upon using the `setuid` test in the p5-test suite, we should see that the `uid` of the process has taken on the `uid` of the file in the output from the `testsetuid` function. In other words, unless the `uid` of the process and the `uid` of the file already match, the `uid` of the process before executing `testsetuid` should be different than the `uid` of the process during execution of `testsetuid` which will be shown in its output.

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

Testing the set uid bit.

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

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

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

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

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

Figure 28: setuid test

As we can see in Figure 28 above, the test passed in all scenarios which includes the test that is designed to fail (the last one shown). We can also see that, when the uid of the process isn't the same as the uid of the file, it takes on the file's uid value in all cases where the file has execute permissions and the setuid bit for the mode is a 1 (all cases in this instance). We can see this because the uid for the process is different before execution than it is during execution, in the output for testsetuid, which matches the files uid before execution. The setuid test PASSES.