**Requests:**

- mkdir() - creates a new directory

- creat() - creates a new (empty) file

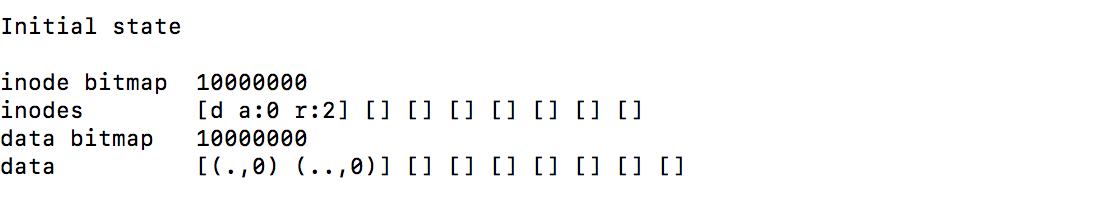
- open(), write(), close() - appends a block to a file

- link() - creates a hard link to a file

- unlink() - unlinks a file (removing it if linkcnt==0)

***1. Run the simulator with some different random seeds (say 17, 18, 19, 20), and  
see if you can figure out which operations must have taken place between  
each state change.***

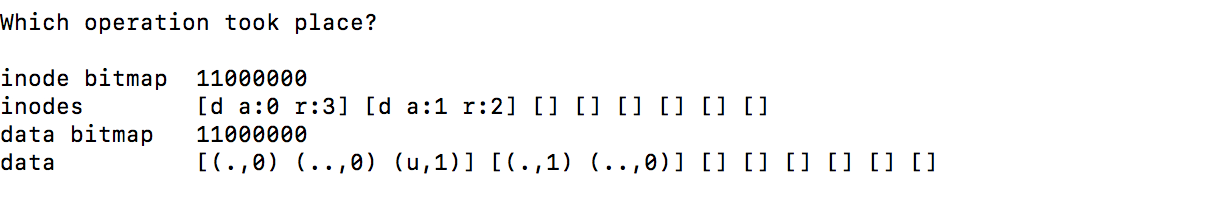
The following contains 10 state changes by running seed 17. It contains all unique requests we need to explain. mkdir(),creat(),open(), write(), close(),link(),unlink():



Initial State:

Inode block 0 is filled with information of root directory.

Data block 0 is filled with a link to inode block 0.



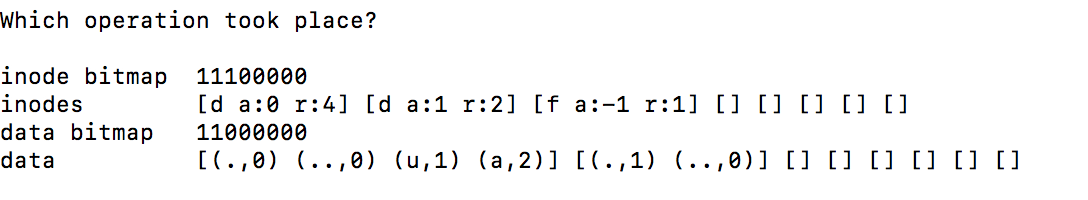
State change 1:

inode block 1 is filled with a new directory file

data block 0 has a file named "u", with a link to inode block 1, appended to it

data block 1 is filled with a new directory

conclusion: a new directory file named "u" was created within the root directory, i.e. **mkdir("/u");**



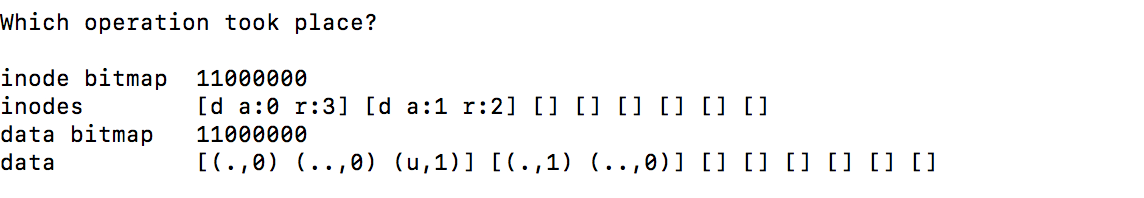
State change 2:

inode block 2 is filled with a new standard file file

data block 1 has a file named "a", with a link to inode block 2, appended to it

data block 2 remains empty as the file "a" created is still empty

conclusion: a standard file named "a" was created within the root directory, i.e. **create("/a");**

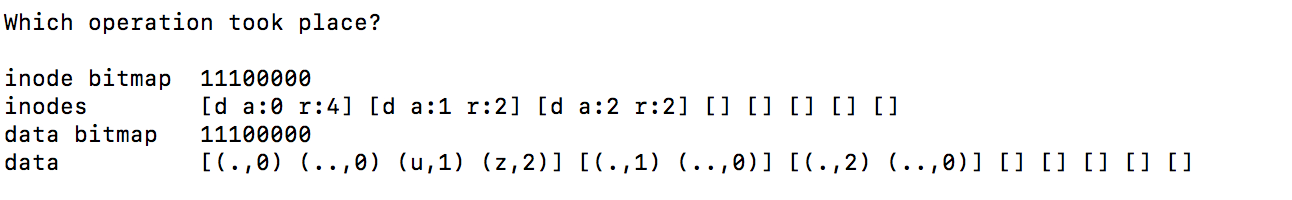


State change 3:

inode block 2 (previously an empty file) is made empty

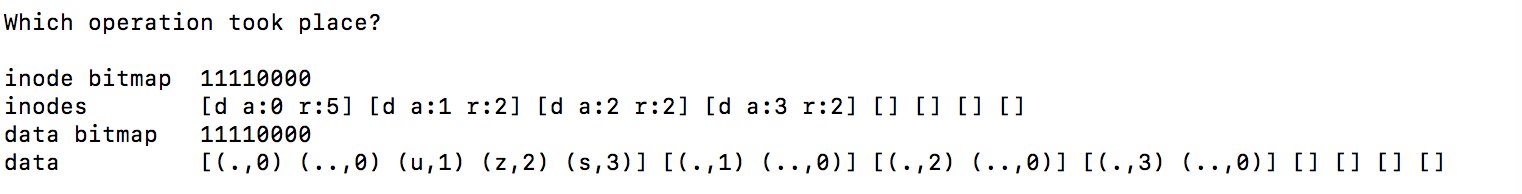
data block 0 (previously containing a file (a, 2)) no longer has (a, 2) in it

conclusion: file "a" lost all 1 of the references to its existence, i.e. **unlink("/a");**

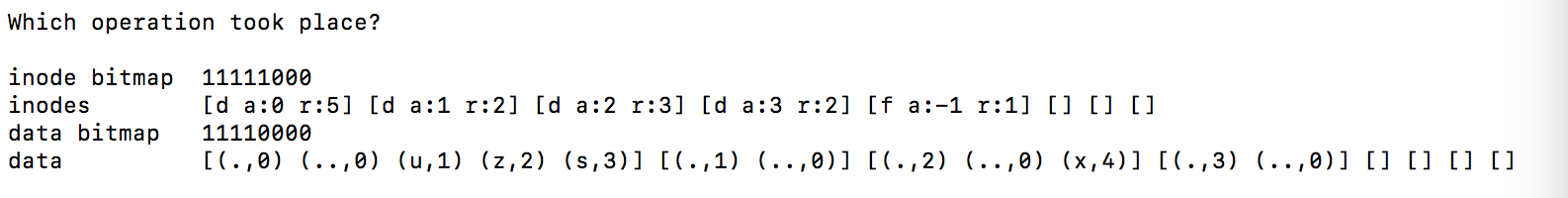


State change 4:

this state change resembles that of state change 1: filling inode block 2 with a directory and appending data block 0 with a file named "z", i.e. **mkdir("/z");**

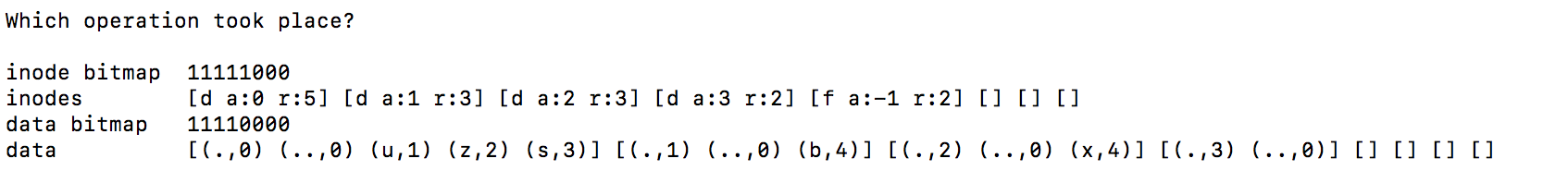


State change 5:  
resembling state change 1 and 4 filling inode block 3 with a dir. file and appending data block 0 with a file named "s", i.e. **mkdir("/s");**



State change 6:

resembling state change 2: filling inode block 4 with a f. file and appending data block 1 with a file named "x", i.e. **mkdir("/z/x");**

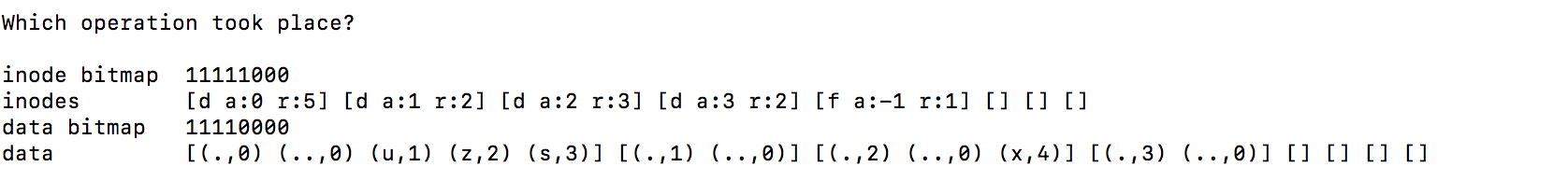


State change 7:

inode block 4 (previously containing a f. file) gains a reference making r = 2

data block 1 (previously an empty directory) has a file (b, 4) appended to it

conclusion: data block 1 gaining a file that references inode block 4, as well as no new f. file being added to the inode blocks, could only mean that a new link to the previously created file has been made, i.e. **link("/z/x", "/u/b");** where /z/x is the f file in inode block 4 and /u/b is the file appended to data block 1.

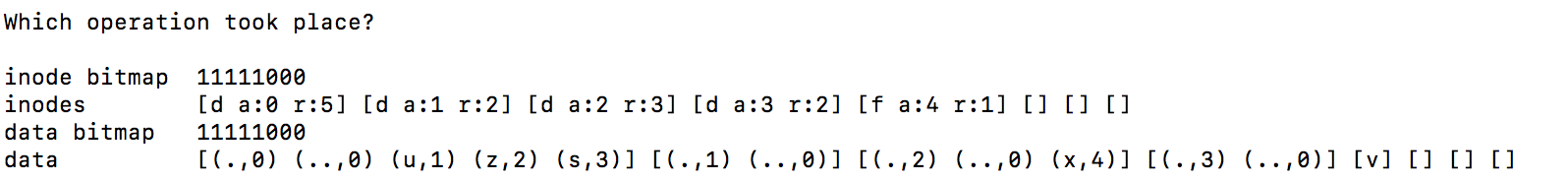


State change 8:

inode block 4 containing a file (of which from state changes 7 and 6 we know is referenced to by paths /z/x and /u/b) loses a reference, r = 1

data block 2 (previously containing a file (b, 4)) no longer has (b, 4) in it

conclusion: these two changes resemble state change 3 except that only file "/z/x" lost only one of its references thus the inode block 4 still has a file in it, i.e **unlink("u/b");**

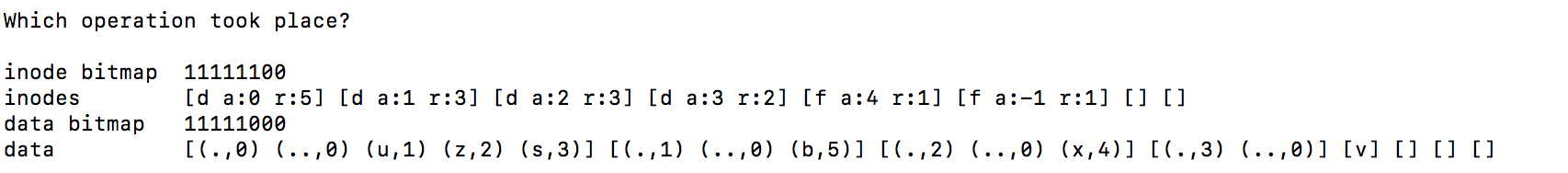


State change 9:

inode block 4 containing a file gains an an allocated data block a = 4

data block 4 then is filled with a single character "v"

conclusion: according to the README data blocks are only seen with a character in them when an f. file is not empty, and since inode block 4 (containing file "/z/x") was previously empty it could only be made non-empty when a write is involved, i.e. **fd=open("/z/x", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

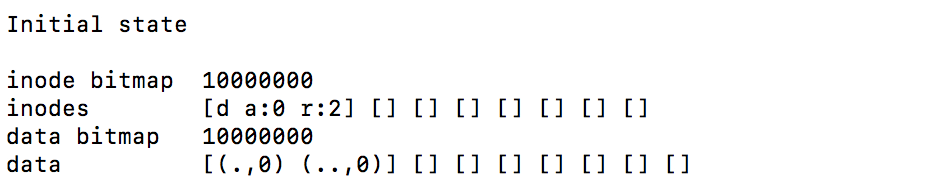


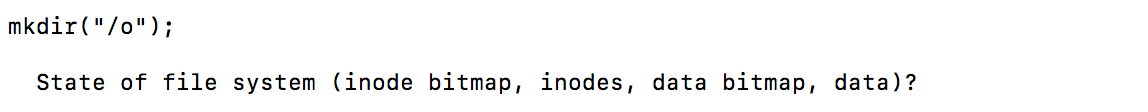
State change 10:

resembles state change 2, as inode block 5 (previously empty) is filled with a f. file and data block 1 with appended with a file named "b", (b, 5), this is almost the same as state change 7 except that a previously empty inode block is filled thus meaning that this new file is not a link to a previously made file but a new empty file all together, i.e. **creat("/u/b");**

***2. Now do the same, using different random seeds (say 21, 22, 23, 24), except  
run with the -r flag, thus making you guess the state change while being  
shown the operation.***

Here is 8 requests for seeds 21. It contains all requests we need to explain:





inode bitmap 11000000

inodes [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []

data bitmap 11000000

data [(.,0) (..,0) (o,1)] [(.,1) (..,0)] [] [] [] [] [] []

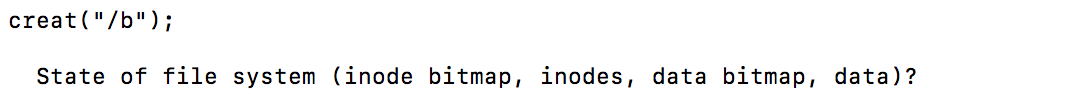
Directory o is created, and it’s added to root directory, and it creates an link to inode 0.

In inode 0, it has one more reference to data block 0.

In data block 0, there is a link to inode 1.

For inode 1, data block 1 is allocated to it.

Data block 1 is filled with two directory.



inode bitmap 11100000

inodes [d a:0 r:4] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []

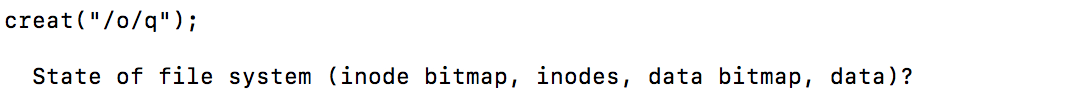
data bitmap 11000000

data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0)] [] [] [] [] [] []

File b is created, and its data is filled in block 0.

For inode 0, the reference number is changed from 3 to 4.

For inode 2, it contains information for file b. File b is linked to it.



inode bitmap 11110000

inodes [d a:0 r:4] [d a:1 r:3] [f a:-1 r:1] [f a:-1 r:1] [] [] [] []

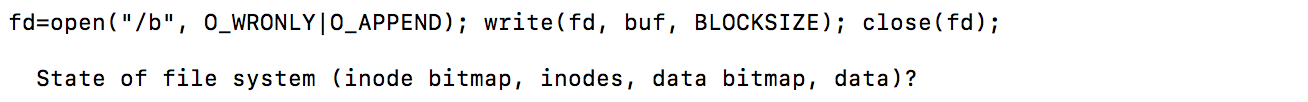
data bitmap 11000000

data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [] [] [] [] [] []

File q is created on o directory, and its data is filled in block 1.

For inode 1, the reference number is changed from 2 to 3. (Inode 1 contains information for data block 1)

For inode 3, it contains information for file q. File q is linked to it.



inode bitmap 11110000

inodes [d a:0 r:4] [d a:1 r:3] [f a:2 r:1] [f a:-1 r:1] [] [] [] []

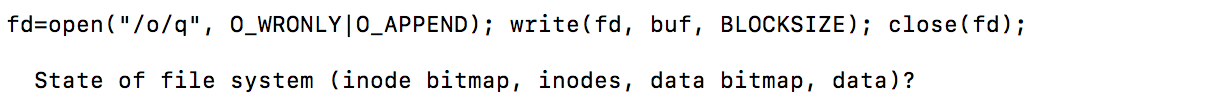
data bitmap 11100000

data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [m] [] [] [] [] []

inode block 2 containing a file gains an an allocated data block a = 2

data block 2 then is filled with a single character "m"

conclusion: according to the README data blocks are only seen with a character in them when an f. file is not empty, and since inode block 2 (containing file "/b") was previously empty it could only be made non-empty when a write is involved,



inode bitmap 11110000

inodes [d a:0 r:4] [d a:1 r:3] [f a:2 r:1] [f a:3 r:1] [] [] [] []

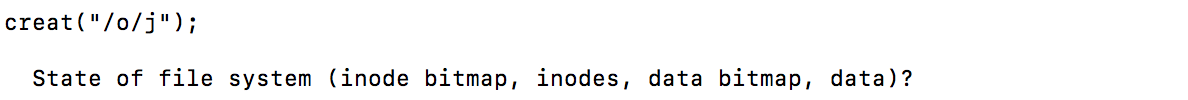
data bitmap 11110000

data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [m] [j] [] [] [] []

inode block 3 containing a file gains an an allocated data block a = 3

data block 3 then is filled with a single character "j"

conclusion: according to the README data blocks are only seen with a character in them when an f. file is not empty, and since inode block 2 (containing file "/o/q”) was previously empty it could only be made non-empty when a write is involved,



inode bitmap 11111000

inodes [d a:0 r:4] [d a:1 r:4] [f a:2 r:1] [f a:3 r:1] [f a:-1 r:1] [] [] []

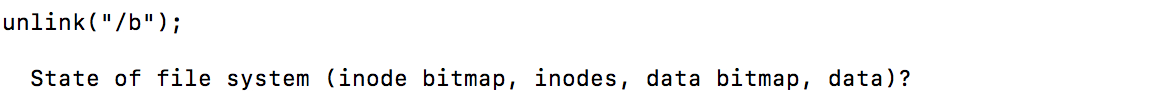
data bitmap 11110000

data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3) (j,4)] [m] [j] [] [] [] []

File j is created on o directory, and its data is filled in block 1.

For inode 1, the reference number is changed from 3 to 4. (Inode 1 contains information for data block 1)

For inode 4, it contains information for file j. File j is linked to it.



inode bitmap 11011000

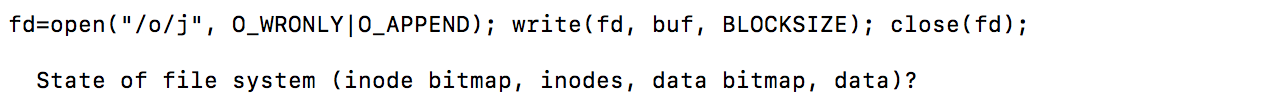
inodes [d a:0 r:3] [d a:1 r:4] [] [f a:3 r:1] [f a:-1 r:1] [] [] []

data bitmap 11010000

data [(.,0) (..,0) (o,1)] [(.,1) (..,0) (q,3) (j,4)] [] [j] [] [] [] []

File b is deleted so (b,2) is removed from data block 0.

For inodes 0, reference number is changed from 4 to 3 since one of file/directory is deleted from data block 0.



inode bitmap 11011000

inodes [d a:0 r:3] [d a:1 r:4] [] [f a:3 r:1] [f a:2 r:1] [] [] []

data bitmap 11110000

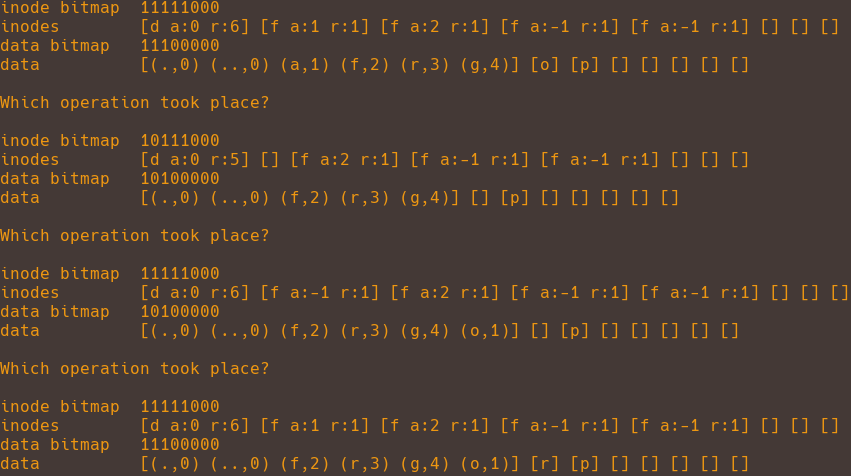
data [(.,0) (..,0) (o,1)] [(.,1) (..,0) (q,3) (j,4)] [g] [j] [] [] [] []

inode block 4 containing a file gains an an allocated data block a = 2

data block 2 then is filled with a single character "g"

conclusion: according to the README data blocks are only seen with a character in them when an f. file is not empty, and since inode block 4 (containing file "/o/j”) was previously empty it could only be made non-empty when a write is involved,

1. ***What can you conclude about the inode and data block  
   allocation algorithms, in terms of which blocks they prefer to allocate?***

The inode blocks and data blocks are always allocated in the left most empty block.   


In this example you will notice the inode and data bitmaps demonstrate the deletion of a file stored in inode block 1 and previously written in data block 1.

bitmap state changes: inode 11111000 -> 1011100, data 11100000 -> 10100000

Upon creating/allocating a new file the leftmost empty inode block is then refilled with the new file.

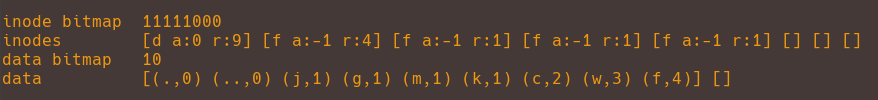
Inode bitmap state change 10111000 -> 11111000

And upon writing to that newly allocated file the leftmost empty data block then refilled.

Data bitmap state change 10100000 -> 111000000

***3. Now reduce the number of data blocks in the file system, to very low numbers  
(say two), and run the simulator for a hundred or so requests.***

1. ***What types of files end up in the file system in this highly-constrained layout?***

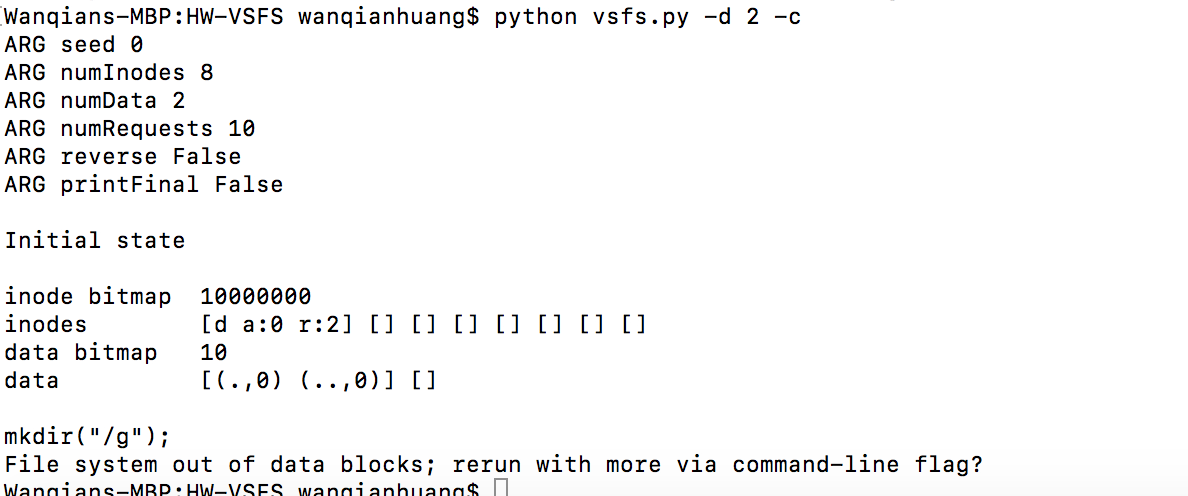
******

Here is a good example of the kind of files that end up in the file system after successful requests with a data block size of 2. Since the only non-deletion related requests involve adding to the inode blocks and appending to the only fillable data block (the root directory) we end up with a lot of empty files (since writing would take up data blocks), in the same directory.

1. ***What types of operations would fail?***

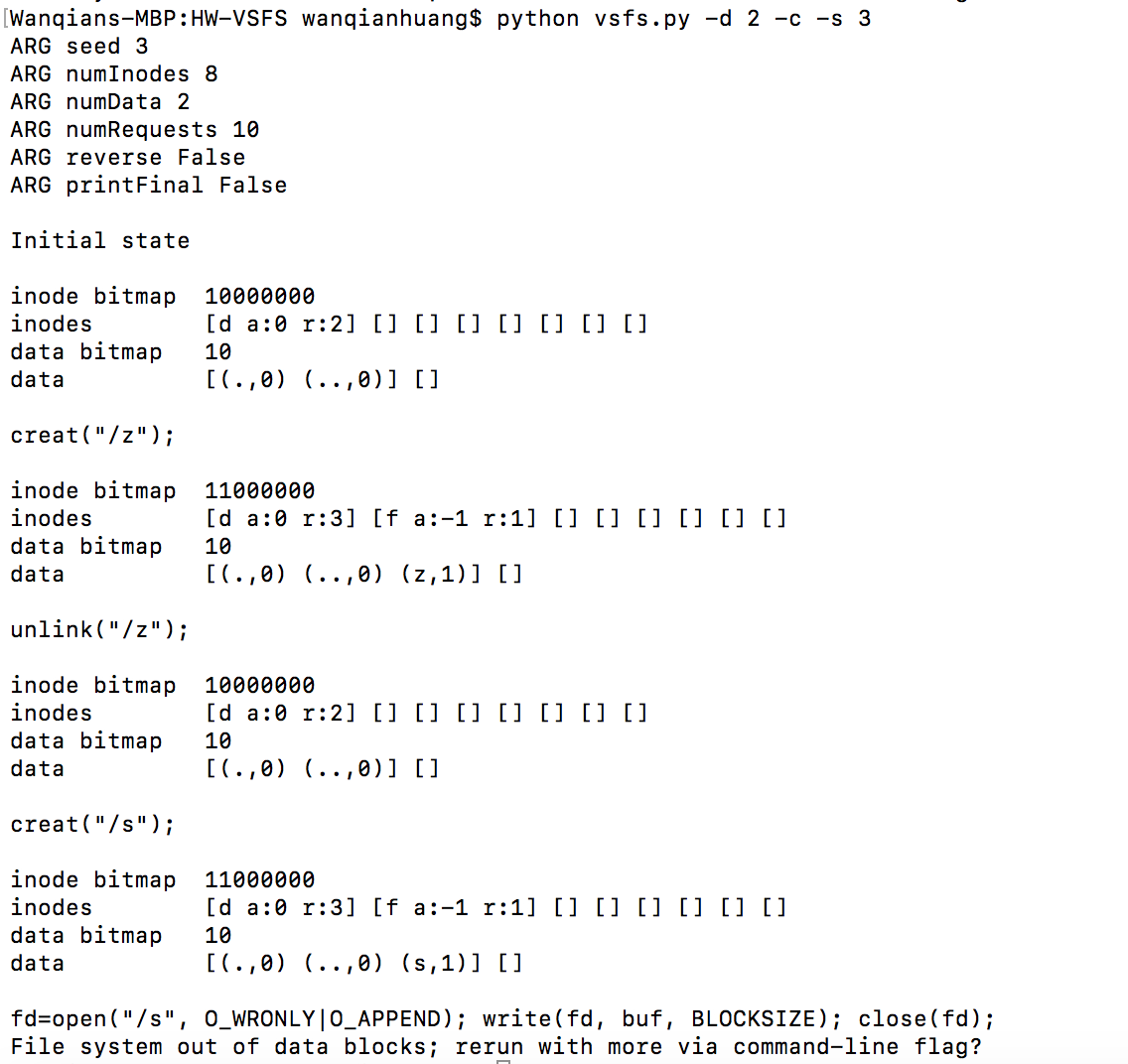
**File system out of data blocks operation failure examples**

**mkdir()** causing a failure: In this example, it has 2 data blocks and 8 inodes. After mkdir request is made, one new data blocks is required for new directory data and one new inode is required for new directory information. Since in this example, it only has 2 data blocks, it may rerun out of data blocks for mkdir request.



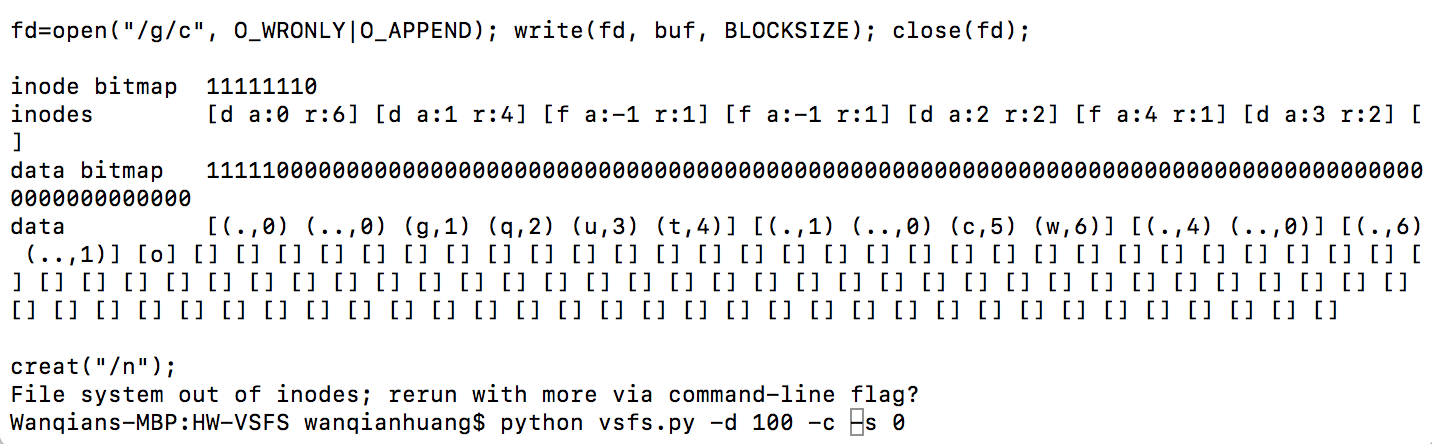
fd = open(); write(); close(fd); Writing to a file causing a failure.

In this example, it has 2 data blocks and 8 inodes. After fd = open();write();close(fd); requests are made, one new data blocks is required for writing information. Since in this example, it only has 2 data blocks, it may rerun out of data blocks for those requests.

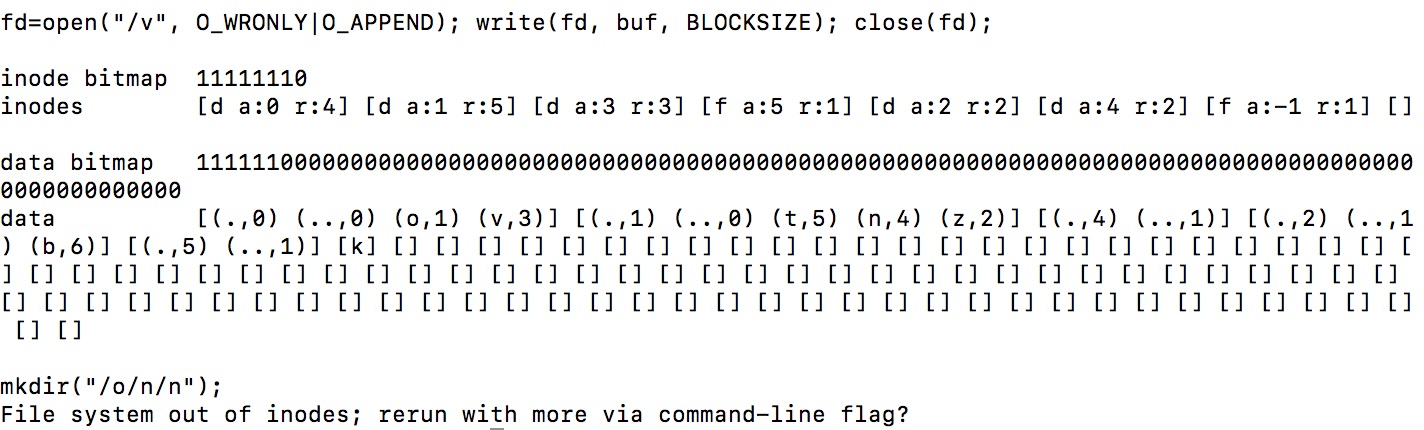


**File system out of inodes failure examples:**

creat() causing failure. In this example, it has 100 data blocks and 8 inodes. After creat request is made, one new inode is required for new directory information. Since in this example, it only has 8 inodes, it may rerun out of inodes for creat request.

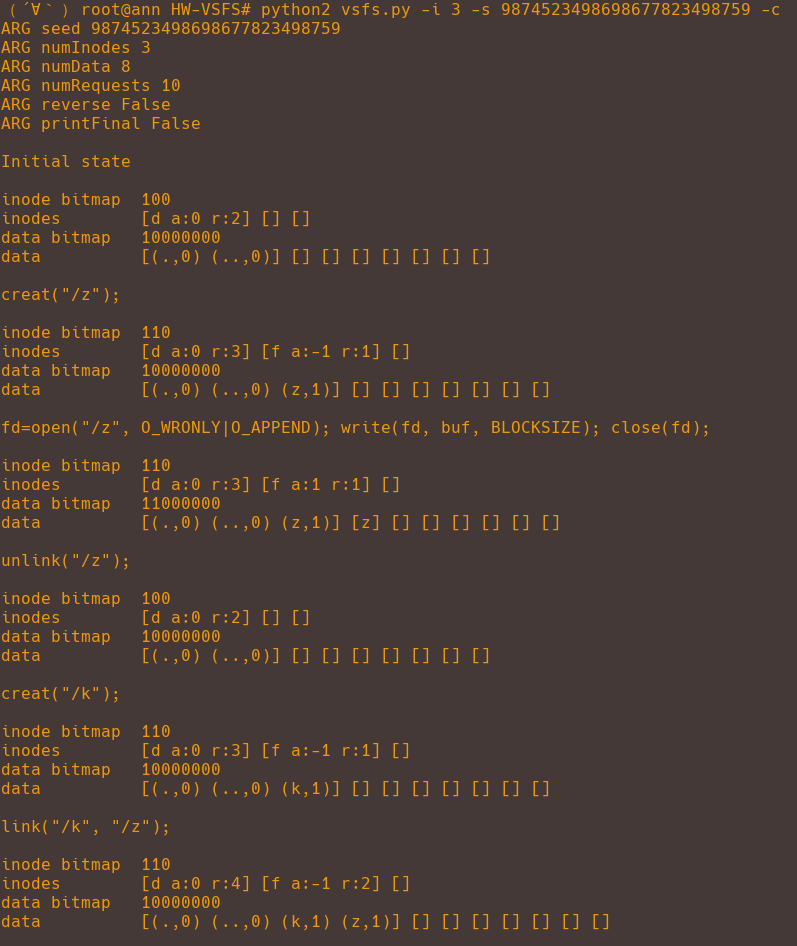


mkdir() causing a failure. mkdir() causing failure. In this example, it has 100 data blocks and 8 inodes. After mkdir request is made, one data blocks is required for new directory data and one inode is required for new directory information. Since in this example, it only has 8 inodes, it may rerun out of inodes for mkdir request.



***4. Now do the same, but with inodes.***

1. ***With very few inodes, what types of operations can succeed?***

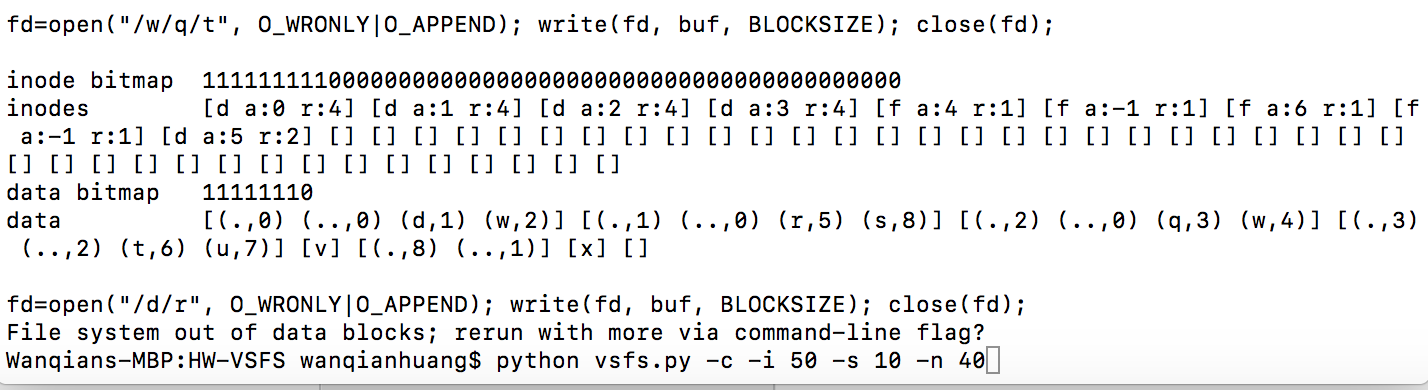
With a min inode count of 3 successful ops are create(), write(), unlink(), link(); these 

1. ***Which will usually fail?***

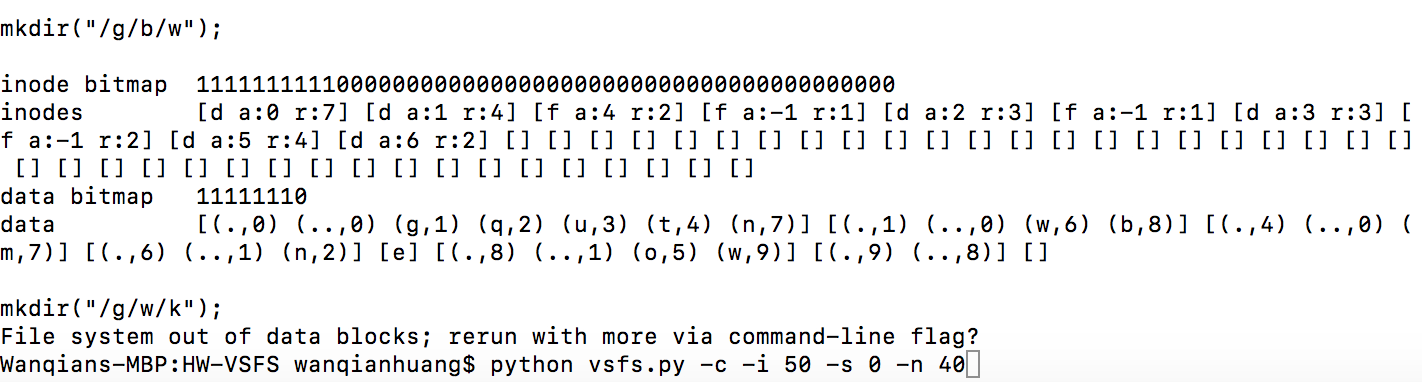
**File system out of data blocks failure examples:**

fd = open(); write(); close(fd); Writing to a file causing a failure.

In this example, it has 8 data blocks and 50 inodes. After fd = open();write();close(fd); requests are made, one new data blocks is required for writing information. Since in this example, it only has 8 data blocks, it may rerun out of data blocks for those requests.

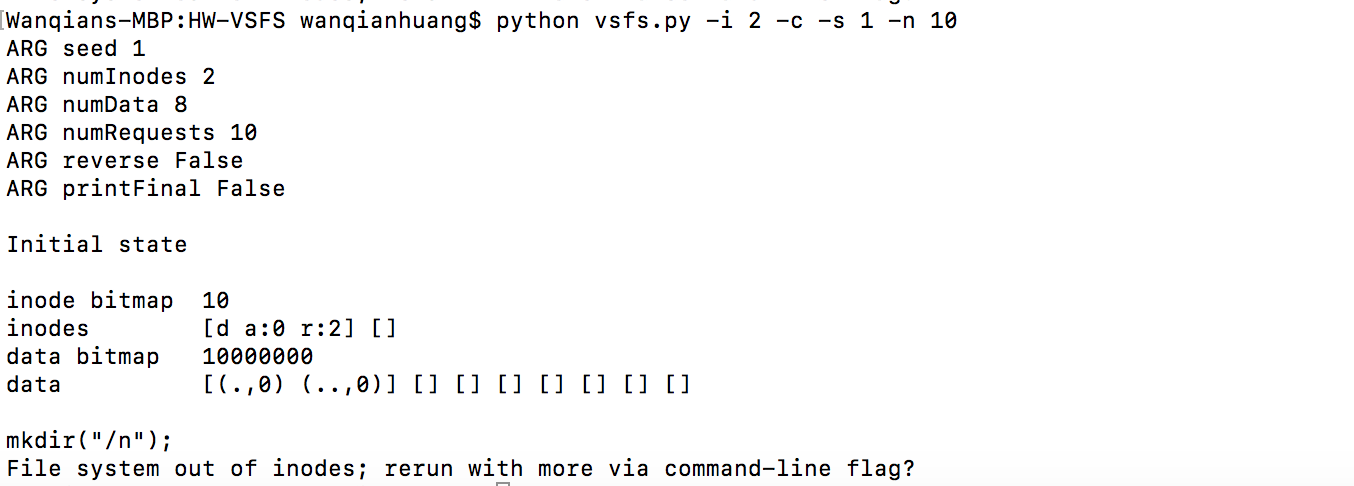


mkdir() causing a failure. mkdir() causing failure. In this example, it has 8 data blocks and 50 inodes. After mkdir request is made, one data blocks is required for new directory data and one inode is required for new directory information. Since in this example, it only has 8 data blocks, it may rerun out of data blocks for mkdir request.

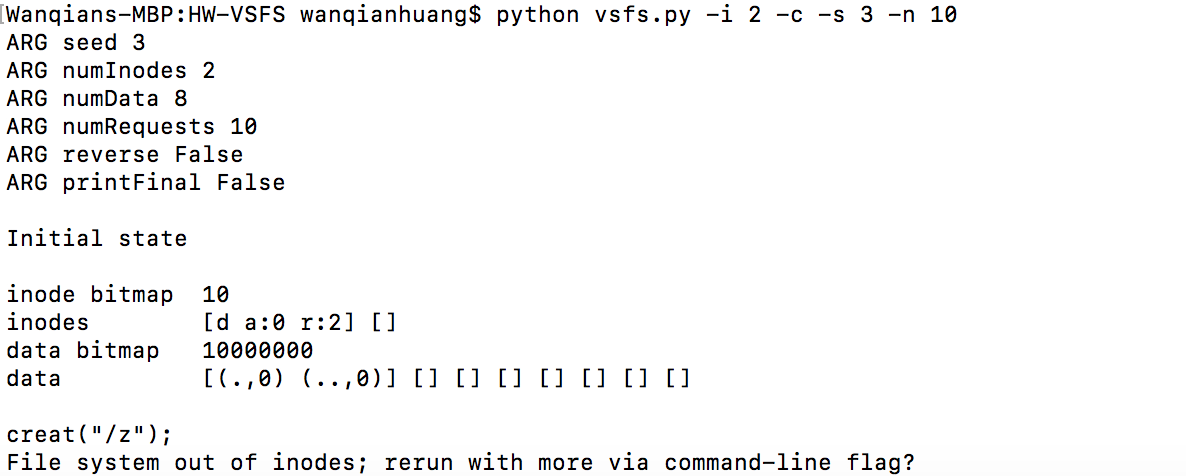


**File system out of inodes failure examples:**

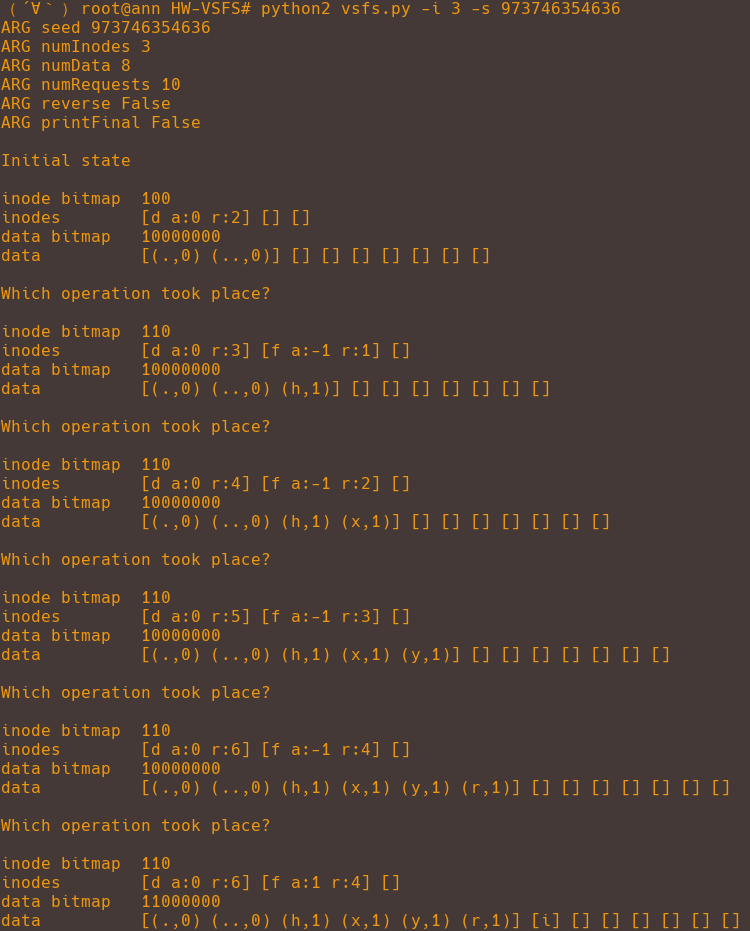
mkdir() causing a failure. mkdir() causing failure. In this example, it has 8 data blocks and 2 inodes. After mkdir request is made, one data blocks is required for new directory data and one inode is required for new directory information. Since in this example, it only has 2 inodes, it may rerun out of inodes for mkdir request.



creat() causing failure. In this example, it has 8 data blocks and 2 inodes. After creat request is made, one new inode is required for new directory information. Since in this example, it only has 2 inodes, it may rerun out of inodes for creat request.



1. ***What is the final state of the file system likely to be?***

******

Here is a good example of the types of files that result in a series of successful requests when the inode block size is only 3. Because the only successful non-deleting requests possible after having filled up the only remaining inode block involve changing the state of data blocks (writing the file, and linking to the file), the final file ends up having many different links to it, and having been written.