Crash Consistency: FSCK and Journaling

Annalise Tarhan

March 16, 2021

1 First, run fsck.py and see if you can determine which files and directories are in there. Try this for a few different randomly-generated file systems.

Seed 1
/m/m /m/e(symlink to /g)
/a/r /a/w(symlink to /g)
/g

Seed 2
/c/o/u/q /c/o/u/e /c/o/q(symlink to /c/o/u/q)

Seed 3
/r/s/
/f /x(symlink to /f)

2 Now, introduce a corruption. Can you find it? How would you fix it in a real file system repair tool?

Seed 1
In the inode bitmap, bit 13 is zero but there is a file inode present and there is a reference to the still unwritten file in the root directory. The solution is to change the inode bit to 1.

3 Change the seed to 3 or 19; which inconsistency do you see? What is different in these two cases?

Seed 3
The inode for file /g/s shows two references, but it is only referenced once.

Seed 19
The inode for directory g only shows one reference, which is impossible for a directory, since it must have a reference to itself and its parent must have a reference as well. Unlike the previous situation, no further examination is needed to detect this corruption.

4 Change the seed to 5; which inconsistency do you see? How hard would it be to fix this problem in an automatic way? Introduce a similar inconsistency with seed 38; is this harder/possible to detect? Finally, use seed 642; is this inconsistency detectable? If so, how would you fix the file system?

Seed 5
The difference between this version and the others is that one of the files has been renamed. There is no internal inconsistency, so the only way to detect the problem would be to compare it to an earlier version.

Seed 38
The problem is detectable because it violates a file system invariant: the first two files must be . and .., a self reference and a reference to the parent directory.

Seed 642
The corruption is also detectable, because it caused two files in the same directory to have the same name, which shouldn't be possible. Unless the previous name is stored somewhere, it would need to be given a new name. w2?

5 Change the seed to 6 or 13; which inconsistency do you see? What is the difference across these two cases? What should the repair tool do when encountering such a situation?

Seed 6
There is a new inode referring to a directory that does not exist in the data. Nothing points to it, so it should simply be deleted.

Seed 13
This is the same situation except that the inode refers to a file instead of a directory. It doesn't point to an allocated block and nothing in the data points to it, so the repair tool should delete it, as before.

6 Change the seed to 9; which inconsistency do you see? Which piece of information should a check-and-repair tool trust in this case?

Seed 9
One of the inodes pointing to a file now claims to point to a directory. A block has not been allocated for it, though, which shouldn't happen if it was really a directory. So, the repair tool should detect the corruption and switch the type back to file.

7 Change the seed to 15; which inconsistency do you see? What can a repair tool do in this case? If no repair is possible, how much data is lost?

Seed 15
Now, the inode of the root directory claims to point to a file instead. Based on the contents of the data block, this is clearly wrong, especially since it refers to the first block, which is always the root directory. The repair tool should change the inode type back to directory. If this somehow wasn't possible, the entire file system's contents would be lost.

8 Change the seed to 10; which inconsistency do you see? Is there redundancy in the file system structure here that can help a repair?

Seed 10
One of the directories' .. file points to an inode that doesn't exist. The fix is to scan the data blocks for a directory with a reference to the corrupted directory's inode. That node will be the parent, and its inode reference should replace the corrupted one.

9 Change the seed to 16 and 20; which inconsistency do you see? How should the repair tool fix the problem?

Seed 16
The inode representing a file without an allocated block has been corrupted to point to an empty block instead. The inode should be changed to point to nothing instead (-1). If it had originally pointed to a block with user data, the data would be lost.

Seed 20
The inode representing a directory now points to an empty block. This can be fixed by scanning the data blocks to find the one whose . file refers to the corrupted inode and making the inode point to that block instead of the empty one.