Homework Wan Huzaifah bin Wan Azhar

Answer:

* 1. Directory: “/”, “/m”, “/a”,
  2. Files: “/g”, “/m/m”, “/e”, “/a/w”, “/a/r”

1. Directory: “/”, “/c”, “/c/o”, “/c/o/u”
2. Files: “/c/o/q”, “/c/o/u/q”, “/c/o/u/e”

* We can see from inode bitmap that the last 3 bit is not set correctly to 1.
* This is the problem of just the data block (Db) is written to disk but its inode bitmap is not correctly set.
* Thus, this can cause issue if file /m and /z is accessed, it will try to get the inode but the inode indicate that there is no data at block 13.



* For -S 3, reference count of /g/s increased to 2 when there is only 1 reference to it.
* For -S 19, reference count of /g decreased from 2 to 1.



* For -S 5 /g/s changed to /g/y
* It a bit difficult to fix this problem in automatic ways. Few approaches:
  + Make backup of the file before writing in the block. Verify using checksum if corrupted.
  + Verify using checksum if corrupted. Remove its inode and inform the user about the error.
  + Use logging system.
* For -S 38 (.., 0) is changed to (b, 0) for /w/b
  + It’s possible to detect it, just check whether the folder has its parent inode. If not, then recreate the parent inode through traversing /w/b.
  + The decision to remove (b,0) is harder to resolve, the system need to know whether (b,0) is symbolic link or parent’s inode.
    - In the former case, the system needs a way to detect if (b, 0) is corrupted data. If it is, then remove it. As the parent inode already added, it won’t be an issue.
    - In the latter case, the system will assume it is a symbolic link to its parent inode, so the system won’t do anything to avoid messing with user data.
* For -S 642, (g,8) is changed to (w,8)
  + The solution is the same as the -S 5 case. It is a data corruption.



* For -S 6, additional directory inode (block 12) is added even though the bitmap are not set and there is no data pointing to the inode.
* For -S 13, additional file inode (block 10) is added even though the bitmap are not set and there is no data pointing to the inode.
* The difference between these two corrupt seed is that -S 6 is directory inode is orphan while -S 13 file inode is orphan.
* The repair tool should just remove the orphan. The system will still maintain consistency and user should just retry if the file is needed.



* Inode 13 [d a:-1 r:2] should be a file, not directory.
* Since the directory is not pointing to anywhere, the system should consider whether it is a file and not a directory.
  + This can backfire, since it might be possible that a=-1 itself is pointing to corrupt address.
  + There is honestly no good way to fix this, can the system really trust that a=-1 refers to file?
  + It is just better to assume that the inode is corrupt and apply corruption fix.



* Inode 0 is a directory, not a file.
* This is a tough case:
  + The repair tool should stop the system and run full system check.
  + It should check the root directory is valid by comparing it to superblock data.
  + After the root directory is valid. The system should traverse the root directory and check the validity of each file and directory and apply necessary fix.
  + In essence, a full check and repair is required.
* If no repair is possible, then all data is lost since there is no way to traverse from root directory by reading the file (Assuming the file has different format than directory format).



* Data block 12 parent directory point to invalid inode 3. (.., 3)
* I have no clue how to use redundancy to fix this.



* For -S 16, inode of file /m and /z point to invalid data address 7.
* For -S 20, inode of directory /g point to invalid data address 11.
* Recovery:
  + For -S 16, try to recover the inode or reset the file address to a=-1 if the first approach is not possible.
  + For -S 20, scan the data block for directory that point itself to /g, fix the inode by pointing to the data address.