CS 111
Computer Systems Principles
Section 1E Week 7

Tengyu Liu

Project 3B – Overview

- In project 3A, input is a filesystem, output is a .csv file
- In project 3B, the input is the .csv file of a corrupted filesystem
- You are asked to analyze and report any corruption you can find
  - Allocated block/inode on the freelist
  - Invalid data blocks
  - Allocated reserved blocks
  - Incorrect links
- You are not required to code in C
  - Higher level languages will make your life easier
  - Python has a csv library
  - Make sure that we can run your program on lnxsrv09

Why would filesystems be corrupted

- Operations to update a filesystem are not atomic
  - A power failure/system crash could cause inconsistency
- Example: deleting a file in a Unix system
  - Step 1: remove its directory entry
  - Step 2: release the inode and add to the pool of free inodes
  - Step 3: release all blocks and add to the pool of free blocks
- What if the system crashes after 1?

Why would filesystems be corrupted

- Operations to update a filesystem are not atomic
  - A power failure/system crash could cause inconsistency
- Example: deleting a file in a Unix system
  - Step 1: remove its directory entry
  - Step 2: release the inode and add to the pool of free inodes
  - Step 3: release all blocks and add to the pool of free blocks
- What if the system crashes after 1?
  - Orphan inode, which creates a storage leak

Why would filesystems be corrupted

- Operations to update a filesystem are not atomic
  - A power failure/system crash could cause inconsistency
- Example: deleting a file in a Unix system
  - Step 1: remove its directory entry
  - Step 2: release the inode and add to the pool of free inodes
  - Step 3: release all blocks and add to the pool of free blocks
- What if the system crashes after 2?

Why would filesystems be corrupted

- Operations to update a filesystem are not atomic
  - A power failure/system crash could cause inconsistency
- Example: deleting a file in a Unix system
  - Step 1: remove its directory entry
  - Step 2: release the inode and add to the pool of free inodes
  - Step 3: release all blocks and add to the pool of free blocks
- What if the system crashes after 2?
  - Blocks used by the files cannot be reused
  - Loss of capacity

Why would filesystems be corrupted

- Operations to update a filesystem are not atomic
  - A power failure/system crash could cause inconsistency
- Example: deleting a file in a Unix system
  - Step 1: remove its directory entry
  - Step 2: release the inode and add to the pool of free inodes
  - Step 3: release all blocks and add to the pool of free blocks
- Can we rearrange the steps to make it safe?
  - Not without creating other problems.

Fixing inconsistencies

- You would have to run a tool such as fsck to find the errors
  - In P3A you produced something similar to debugfs
  - In P3B you will produce something similar to fsck
    - Except fsck fixes the errors, you just have to identify them
- Is there a more modern way of handling inconsistencies?

Fixing inconsistencies

- You would have to run a tool such as fsck to find the errors
  - In P3A you produced something similar to debugfs
  - In P3B you will produce something similar to fsck
    - Except fsck fixes the errors, you just have to identify them
- Is there a more modern way of handling inconsistencies?
  - Journaling
  - A journal is a data structure that records intentions of changes before they are made
  - Changes that were started can be resumed more easily after system crash

Where to start?

- Parse the .csv file to a structure / object
  - The csv library in python makes this process very easy
  - csv.reader() will iterate through lines for you
  - It returns a list of string for each line
  - The first value determines how you would parse the rest of the line
    - If the first value is SUPERBLOCK, expect 7 more values
- You will refer to the .csv file a lot, so better store it in a structure / object.

What are the possible inconsistencies?

- You are asked to report inconsistencies and errors on
  - Inodes
  - Blocks
  - Directories
- We will not corrupt superblock or group descriptor table

Possible inode inconsistencies

- Used and free
  - i_mode != 0 and inode # is free on the bitmap
- Unallocated and in use
  - I_mode == 0 (which means there is no dedicated line in csv)
  - And inode # is marked as in use on the bitmap
- Compare the two sources of information (freelist lines and inode lines) and report inconstancies

Possible block inconsistencies

- Invalid
- Reserved
- Unreferenced and used
- Allocated and free
- Duplicate

Possible block inconsistencies

- Invalid
  - Block # < 0 || block # > max block number
- Reserved
- Unreferenced and used
- Allocated and free
- Duplicate

Possible block inconsistencies

- Invalid
- Reserved
  - Block # is used by boot sector block, superblock, group descriptor table, inode bitmap, block bitmap, or inode table
- Unreferenced and used
- Allocated and free
- Duplicate

Possible block inconsistencies

- Invalid
- Reserved
- Unreferenced and used
  - Block # is not referenced by any inodes, but is marked as allocated on the block bitmap
- Allocated and free
- Duplicate

Possible block inconsistencies

- Invalid
- Reserved
- Unreferenced and used
- Allocated and free
  - Block # is referenced by an inode, but marked as free on the block bitmap
- Duplicate

Possible block inconsistencies

- Invalid
- Reserved
- Unreferenced and used
- Allocated and free
- Duplicate
  - Block is referenced by 2 or more inodes
  - **Duplicate block must generate duplicate reports**

Some useful functions

- Calculating the total number of blocks
  - By looking at the superblock
  - This gives you an upper bound for valid blocks

- Calculating the first legal block number
  - Also from the superblock
  - Gives you a lower bound on legal blocks

Some useful functions

- Calculating the total number of blocks
  - By looking at the superblock
  - This gives you an upper bound for valid blocks

- Calculating the first legal block number
  - Also from the superblock
  - Gives you a lower bound on legal blocks
  - How?

Some useful functions

- Calculating the total number of blocks
  - By looking at the superblock
  - This gives you an upper bound for valid blocks

- Calculating the first legal block number
  - Also from the superblock
  - Gives you a lower bound on legal blocks
  - How?
    - The superblock give you first inode block
    - Need to add the sizes of the inode table

Block references: the idea

- Initialize some state variable for blocks
  - Can be free, reserved, in use or duplicate
- Extract bitmap information
  - By looking at all bfree lines
- Create a second bitmap structure
  - Initialize it to be empty, then fill it up by looking to the inode lines of the csv
- Compare both bitmaps

Possible directory inconsistencies

- Incorrect link count
- Unallocated
- Invalid
- . is not self
- .. is not parent

Possible directory inconsistencies

- Incorrect link count
  - # of entries pointing to inode does not match inode link count
- Unallocated
  - i_node referenced in entry is marked as free on bitmap
- Invalid
  - i_node # referenced in entry is invalid
- . is not self
- .. is not parent

Some useful structures

- A structure to keep track of each inode's parent number
  - You will fill it up as you go through directories
- A structure in which you count all references to inodes
  - You will increment the counter for each inode as you go through directories
  - You will be able to compare that information to what was reported in the csv file

Questions?