**CS551 Project3 – Design Document**

**File System Recovery**

**Date: 26th April 2017**

**Primary Author(s): Group8 - Chethan Bhaskar, Shashank Ravindranath, Sudhindra Manohar Bellur Vasudeva**

***Abstract:***

This document describes the basic concept of Minix file system followed by design and implementation of tools DirectoryWalker, iNodeBitMapWalker, ZoneBitMapWalker that are used in damaging the file system and recovering from the damage. The document also explains the new system calls implemented to achieve the goal.

Table of Contents

[Manual page of the tools 2](#_Toc481006059)

[Directorywalker 2](#_Toc481006060)

[InodebitMapWalker 2](#_Toc481006061)

[ZoneBitMapWalker 2](#_Toc481006062)

[Design of the Tools 3](#_Toc481006063)

[DAMAGE TOOL– implementation and usage 3](#_Toc481006064)

[RECOVER TOOL – implemenatation and usage 4](#_Toc481006065)

[The inode bit map is damaged 4](#_Toc481006066)

[The Zone Bit map is damaged 4](#_Toc481006067)

[Other type of damages is handled 4](#_Toc481006068)

[Exception Handling 5](#_Toc481006069)

[Recover tool - When device name is not given to recovery tool, it exits gracefully by showing the correct usage. 5](#_Toc481006070)

[Recover tool - When a wrong/nonexistent device name is provided it exits gracefully. 5](#_Toc481006071)

[Recover tool - When more than required arguments are provided, tool exits gracefully. 5](#_Toc481006072)

[Damage tool – when the file name doesnot exists 5](#_Toc481006073)

[How recovery is done 6](#_Toc481006074)

[Part of the superblock is damaged 6](#_Toc481006075)

[The inode of a file is damaged 6](#_Toc481006076)

[A block (or blocks) allocated to a file is damages 6](#_Toc481006077)

[Extra Features: 7](#_Toc481006078)

# **Manual page of the tools**

All the tools that we have designed are basically used to recover from the damage happened to the file sytem.

We have created two testing tools to test these requirements. One of them is recovery tool. In the recovery tool it uses the same functionality as fsck command in minix system. We have used open(), close(), read() and write() system calls are the building blocks of the recovery tool. Using this file descriptor which points to root of the file system of the given device, tool does read() and write() calls to read, write data of blocksize from disk.

All the tools that are developed below doesn’t have a user interface, instead we have implemented as part of the recovery tool that is used for testing. So all the walkers are used internally, called internally and displays whenever each event happens while performing the recovery of damaged file/directory.

## **Directorywalker**

We have used the system calls that are mentioned above to walkthrough the directory trees in the system. This also uses the below mentioned implementation to walk through the directory. This displays the all the files that are in the current directory and also list the corresponding inode and zone number of these files.

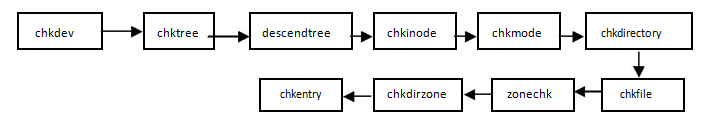
## **InodebitMapWalker**

We have used the above listed system calls to walk through the inodes in the system­­. We have written all these system calls in our file and used the sytem calls to perform the action. Chktree starts from the root node and starts the check through descendtree. This method takes the directory node and checks its inode. Then chkinode checks the inode, takes the inode and uses the chkmode to check for links, symbolic links, and directories separetly. Thus, traversing over the file system tree from its root, we get the inodes, files list currently available in the system. As we can see in this implementation of the tool, it traverses the file system tree from its root (this is the **Directory walker tool**) it lists all the files, inodes and zones that are present in the current directory. As it travesers the inode of the directory it uses the **InodeBitMapWalker.**

## **ZoneBitMapWalker**

In this tool also, the functionality is achieved by using the existing system calls but, written in our custom file. **ZonneBitMap** is fethed by reading from the disk. This is performed by using the Super Block, we can calculate the location of the zone bit map on the disk. This bit map is cross checked across the zone bitmap that is calculated from the indoes list. If there is any inconsistency then, the zonemap present on the disk is overwritten with the one that is calculated using inodes.

# **Design of the Tools**



The above diagram describes the flow of all the tools that we have implemented and how it finds the inconsistency in the system. The main method is process the options and sets the flag. Then it takes the device and does a chkdev call to check the device.

Then the method chkdev,

* It reads the Super Block, check for the inconsistency
* The flow does a chktree, starts from the root node and starts the check through the descendtree.
* Descendtree method takes the directory node, and checks for the flags and checks the inode.
* The call to cknode takes the inode, and uses chkmode to check for links, symbolic links and directories separately.
* The method chkdirectory checks the individual files in the directory using the call to chkfile.
* This method starts checking the zones, which in turn call the zonechk method and checks the zone entries
* Then checks the direct zones chkdirzone then checks individual entries.

Now let’s see how the testing tools have been implemented and how these use the above tools internally. Their design, implementation and usage.

# **DAMAGE TOOL– implementation and usage**

We have used different methods to damage the file system and use the recovery tool to repair the damaged ones. Some of the ways that we used to damage the file system is as follows.

We have created a tool called damageFileSystemtool, tool to damage the filesystem. We have created two new system calls to do this – customlink() and customunlink(). We have provided the menu to do the different damages to the file system. This menu is created based on the requirement given.

# ./damageFileSystemTool

DAMAGE FILE SYSTEM TOOL – please select your choice

1. Delete the file without damaging the folder
2. Damage the inode bit Map by removing the file.
3. Currupt the Directory file completely.
4. Damage the inode time to damage the inode bit map.
5. Damage the zone bit map.
6. Damage the Directory file by corrupting its inode completely
7. Exit

Enter your choice ->

# **RECOVER TOOL – implemenatation and usage**

The recovery tool recoverFileSystemTool functionality is same as the fsck command that we use in the minix.

# ./recoverFileSystemTool

Invalid number of arguments.

Usage: ./recoverFileSystemTool <device-name>

Example: ./recoverFileSystemTool /dev/c0d0p0s0.

How recover is performed on different type of damages or corruption we have done on the file system.

## **The inode bit map is damaged**

When the inode map is damaged, it can be recovered by traversing the file system tree i.e, visiting each file, directory in the disk and noting their inode numbers. This list of inode numbers indicate the inodes that are currently in use. With this list in hand one can calculate the inode bitmap.

## **The Zone Bit map is damaged**

When zone map is damaged, it can be recovered by going over the inode map and checking the zones that are used by each inode. When one knows the zones that are used by all the inodes in the system, zone bitmap can be easily reconstructed by using this list of zones that are currently used by inodes.

## **Other type of damages is handled**

* If the entry of the file in the directory gets damaged but the file is still present then, upon recovery the entry of the file in directory is removed and the file is deleted.
* If the file in the directory gets damaged and the entry of the file is still in the directory, then upon recovery the entry of the damaged file is removed from the directory
* If the file access time i.e. the inode of the file gets damaged, it can be recovered to a closet time like file last modified or inode access time (or edit the file to recover).
* If the directory inode (link) gets damaged then to do recovery necessary matching is done with the directory and the information updated and recovery is done

# **Exception Handling**

## **Recover tool - When device name is not given to recovery tool, it exits gracefully by showing the correct usage.**

# ./recoverFileSystemTool

Invalid Number of arguments.

Usage: ./recoverFileSystemTool <device-name>

Example: ./recoverFileSystemTool /dev/c0d0p0s0

for device name execute command df

## **Recover tool - When a wrong/nonexistent device name is provided it exits gracefully.**

# ./recoverFileSystemTool abc

abc: no such file or directory

couldn't open device to fsck fatal

## **Recover tool - When more than required arguments are provided, tool exits gracefully.**

# ./recoverFileSystemTool /dev/c0d0p0s0 abc

Invalid Number of arguments.

Usage: ./rfstool <device-name>

Example: ./rfstool /dev/c0d0p0s0

for device name execute command df

## **Damage tool – when the file name doesnot exists**

# ./damageFileSystemTool

DAMAGE FILE SYSTEM TOOL – please select your choice

1. Delete the file without damaging the folder
2. Damage the inode bit Map by removing the file.
3. Currupt the Directory file completely.
4. Damage the inode time to damage the inode bit map.
5. Damage the zone bit map.
6. Damage the Directory file by corrupting its inode completely
7. Exit

Enter your choice -> 0

\*\*\*\*\*\*Enter file name along with the path\*\*\*\*\*

Enter absolute file path: abc

Name = abc

damageFileSystemTool: abc: no such file or directory.

# **How recovery is done**

## **Part of the superblock is damaged**

EXTERN struct super\_block {

ino\_t s\_ninodes; /\* # usable inodes on the minor device \*/

zone1\_t s\_nzones; /\* total device size, including bit maps etc \*/

short s\_imap\_blocks; /\* # of blocks used by inode bit map \*/

short s\_zmap\_blocks; /\* # of blocks used by zone bit map \*/

zone1\_t s\_firstdatazone\_old; /\* number of first data zone (small) \*/

short s\_log\_zone\_size; /\* log2 of blocks/zone \*/

unsigned short s\_flags; /\* FS state flags \*/

off\_t s\_max\_size; /\* maximum file size on this device \*/

zone\_t s\_zones; /\* number of zones (replaces s\_nzones in V2) \*/

short s\_magic; /\* magic number to recognize super-blocks \*/

}

The super block has the above structure. If the entries like s\_imap\_blocks, s\_zmap\_blocks are corrupted then they can be recovered/corrected by using the entries like s\_ninodes, s\_zones, block\_size. However, if entries like log\_zone\_size, s\_ninodes, s\_nzones are corrupted then it will be difficult to recover as these can’t be calculated especially log\_zone\_size. A wrong log\_zone\_size will have a catastrophic impact on the system.

## **The inode of a file is damaged**

EXTERN struct inode {

u16\_t i\_mode; /\* file type, protection, etc. \*/

u16\_t i\_nlinks; /\* how many links to this file \*/

u16\_t i\_uid; /\* user id of the file's owner \*/

u16\_t i\_gid; /\* group number \*/

i32\_t i\_size; /\* current file size in bytes \*/

u32\_t i\_atime; /\* time of last access (V2 only) \*/

u32\_t i\_mtime; /\* when was file data last changed \*/

u32\_t i\_ctime; /\* when was inode itself changed (V2 only)\*/

u32\_t i\_zone[V2\_NR\_TZONES]; /\* zone numbers for direct, ind, and dbl ind \*/

}

The above is the inode structure in minix. When an entry like i\_nlinks, i\_size is damaged then inode can be recovered. For ex: a rough value for i\_size can be calculated by checking the number of i\_zones that are used by this inode. If i\_zone array is corrupted then it can’t be recovered as the allocation of zones in minix is not contiguous.

## **A block (or blocks) allocated to a file is damages**

When a block (or blocks) of a files damages then it is difficult to recover the complete file as some block(s) are completely damaged. However, the files can be partially recovered.

# Extra Features:

1. In damage tool, there is a provision to set last access and last modified times of a file to 0. Thus a way to corrupt the files inode.
2. When any files last access and last modified times in its inode on the device is zero user is provided with an option whether he/she intends to delete the file.
3. In recovery tool, when a directories inode has last access and last modified times as zero then an option is provide whether to delete all files under that directory and directory or not.
4. Using damage tool, one can set last access and last modified times of a directory to zero.