CS450 PA4

12-02-2021

Isaias Rivera
Dave Anderson

1)

Description of design :

The design of a directory tree recovery system initially requires the implementation of both a program that can take account of the directory structure stored in memory and a program that can take account of the directory structure written to the disc. Developing two system calls that can accomplish this is naturally the basis for creating tools that can repair information if the disc is damaged. getdiNode() is an existing xv6 function that can be used to iteratively retrieve data from the inode table the disc, where dinode is the on disc equivalent of an inode in memory. This particular function call makes an inode table walker slightly more simple to implement than the directory walker. The output of a directory walker should resemble the output of a Linux 'tree' command. The 'ls' command is useful in developing a directory walker. In order to print the tree, the directory walker recursively traverses the full directory structure listing the contents of each directory along the way. Calling the stat function is useful for retrieving the inode and size information.

With an inode table walker and a directory walker in place, a recovery walker can then be developed. However there must first be a file system, which also becomes damaged, for there to be any 'recovery'. The test file is used to create the test file system and then also to simulate damage to a file system. The test file takes advantage of a random generator to build out the tree. The wither() function in test ultimately uses funlink() to simulate the destruction of a dinode. Finally, the recovery walker uses the two established walkers, directory and inode table, and compares the outputs of both. It takes into account the differences caused by the simulated damage, noting the information left undamaged in the memory side of the output pairings, and reverses the action.

The inode table walker, directory walker, test, and recovery walker are implemented in xv6 like other system calls. The Makefile, syscall.c, syscall.h, sysfile.c, usys.S, user.h, defs.h, and fs.c all require edit for the calls to be added correctly.

Man pages for added system calls :

Name:

directoryWalker - walks and prints the directory tree

Synopsis:

directoryWalker [ ]... [starting directory]

Description:

directoryWalker starts at either the root of a directory tree or at the directory argument given to the system call, and displays the directory tree and file system. The inode number and size in memory are printed with directory name or file.

Name:

inodeTBWalker - walks and prints dinode table from the disc

Synopsis:

inodeTBWalker [ ]

Description:

inodeTBWalker iterates through the inode table on disc printing the dinode number, type, and size.

Name:

test - generates and manipulates a directory tree of directories and files of varying sizes

Synopsis:

test [ ]... [directory name]

Description:

test creates a ‘dummy’ directory tree populated with generated files and the directories that they are organized in. test also uses a function called wither( ) which takes the argument for a directory inode to be deleted, making the resulting directory tree ‘withered’ or damaged compared to the original directory tree. With a damaged directory tree in memory, the recoveryWalker system call can be demonstrated.

Name:

recoveryWalker - walks and prints the directory tree

Synopsis:

recoveryWalker [ ]

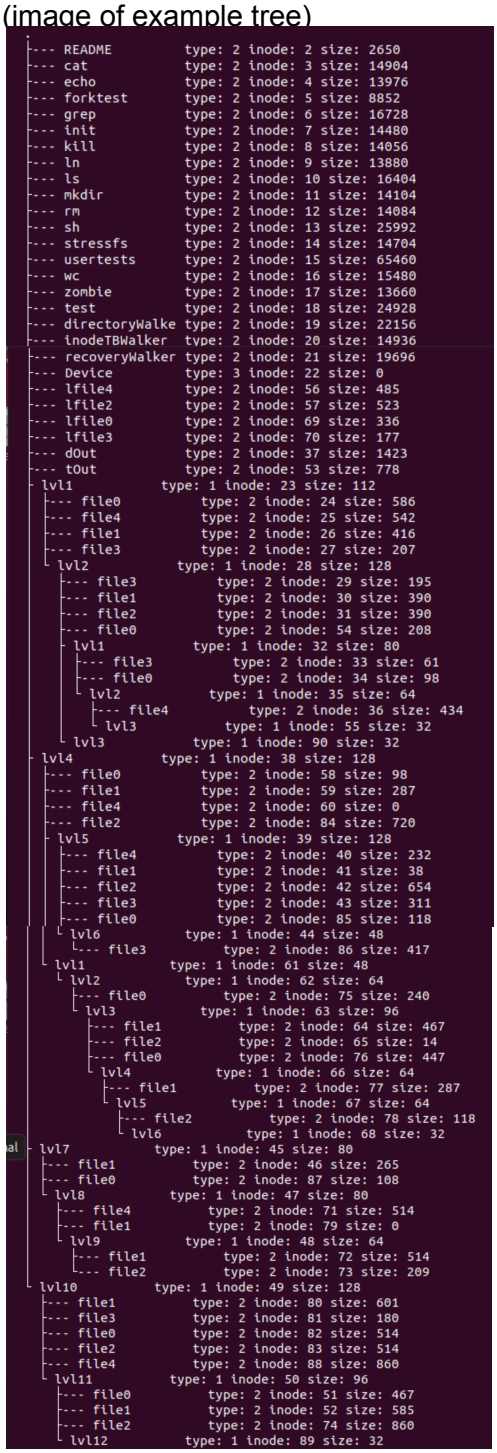
Description:

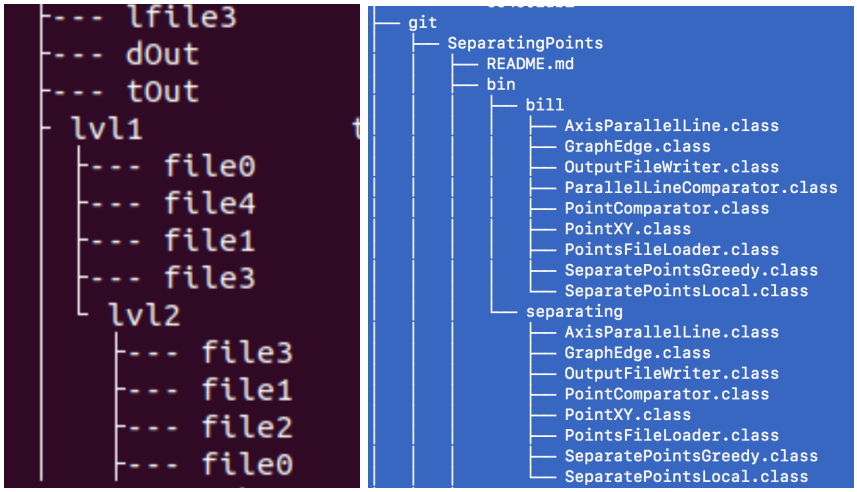
recoveryWalker uses outputs from the directoryWalker and inodeTBWalker system calls to generate a comparison of the two lists. Using a comparison of the list i.e damaged information that is paired with the undamaged data left in memory, recoveryWalker generates repairs to the inode table on disc

2)

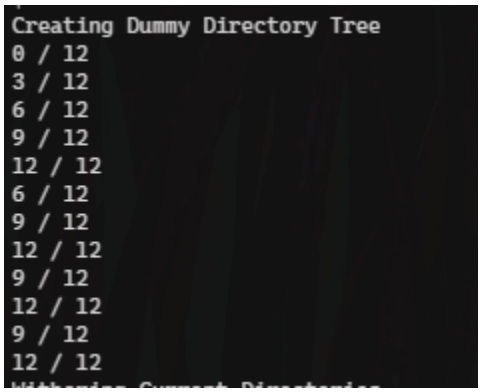
The test program generates a directory structure that can be considered good or useful test data because it creates a directory tree with varying levels of ‘depth’ and asymmetry that resembles the nature of a real file system. The goal of the programming assignment is to simulate the damage of a file system on disk, and to simulate a recovery program that can repair the file system using inode information stored in memory. The best way to develop a test file directory is to create a ‘dummy’ file directory that doesn’t contain any useful files, but that looks similar to and simulates a real file system.

(image of example tree)

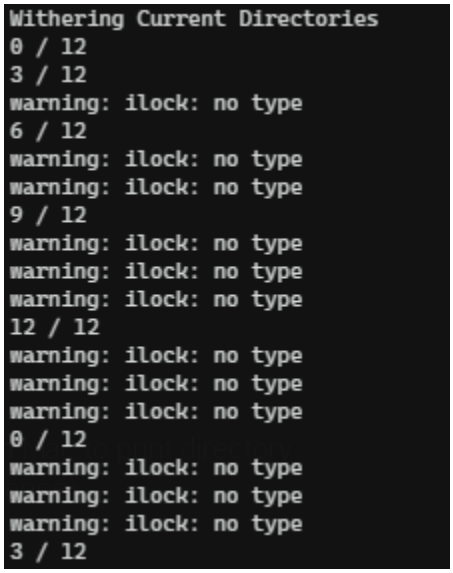




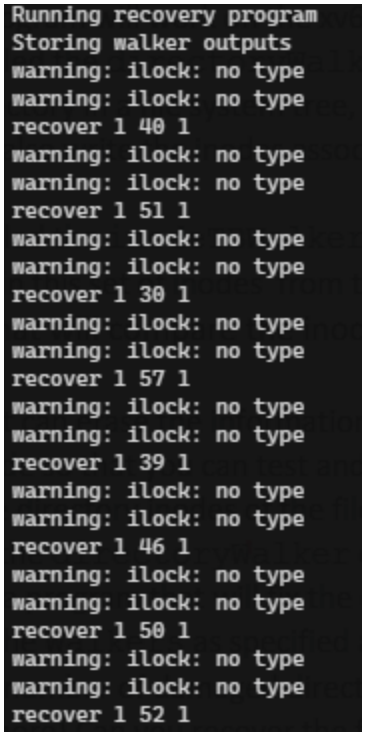
(image of test generated file system structure and ‘tree’ command used on Mac to print directory tree. The test generated file system is a good simulation of the user experience)



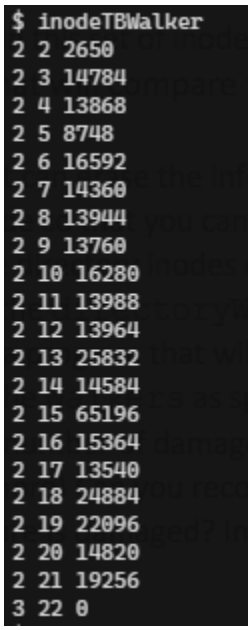
(Image of dummy tree being made)



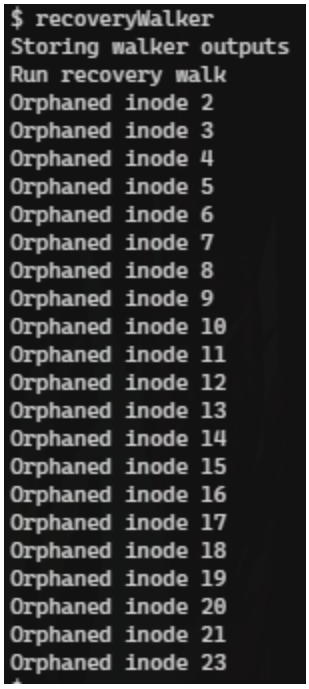
(Image of nodes being damaged, kernel level functions complaining due to the invalid inodes)



(Image of basic recovery done by the directory walker, resets the inode type and makes it valid again)



(image of inodeTBWalker working)



(Image of recoveryWalker comparing the output of both walkers, given no recovery is run)

Changes
Defs.h - headers for fs functions
Fs.c - idget & irec
Makefile - additional programs
Syscall.c, syscall.h user.h usys.S- new system calls