Josiah Hunt (A20350987) and Mayank Bansal (A20392482)

CS 450- Operating Systems (Spring 2018)
Group Programming Assignment - 4

PA4 Write Up

Our design makes use of multiple integer arrays to track allocations and consistency across the walkers. Each walker populates an array the size of the inode table. The index of each array element corresponds to an inode, and if that element is set to 1 then it is allocated. Both the inode walker and the directory walker populate their allocation arrays. These two arrays are then compared by the comparisonWalker, and the results of that comparison are also stored into an integer array. If the allocation array elements are equal, then the array element is set to 0. If there is an inconsistency, the element is set to 1. The recovery walker in turn uses the consistency array populated by the comparison walker to determine which inodes have been damaged, and to re-link them to their parent directory.

Pt. 6: Using these syscalls, we are able to recover exactly one level of directories. The syscall only relinks lost directories to their parent, not their children, so child files are still lost.

We didn't change any existing functions, but implemented several of our own. In fs.c,
formatName(char *path): Makes printing the name look nicer
printIndent(int indent): Makes printing the file hierarchy look nicer
checkDir(): Checks that the directoryWalker allocation array has been populated
checkInode(): Checks that the inodeWalker allocation array has been populated

Created five driver programs
coWalker.c: Runs the compareWalker syscall
daInode.c: Runs damageInode syscall
fsWalker.c: Runs the directoryWalker syscall
inWalker.c: Runs the inodeWalker syscall
reInode.c: Runs the recoverInode syscall

Updated all header files associated with registering a new syscall (sysproc.c, syscall.h, syscall.c, user.h, usys.S, makefile to include test program(s)). Only changes made were those required to register a new system call, just like in PA2.

Manual Pages

Name: Directory Walker
Synopsis

```
#include <user.h>
#include <types.h>
#include <syscall.h>
int directoryWalker(char* path)
```

Description

The directoryWalker() syscall prints out all directory names and files names in the file system tree given a specific starting point. Prints the name and inode number of all files in the base directory, recursively traveling into any child directories. Populates the corresponding elements of an integer array to reflect the allocation status of each file inode. After printing all files and child files, returns.

Return Value

If path is invalid, returns -1

if path is valid, returns 0

Errors

directoryWalker() can fail if specified path is invalid.

If a directory inode is damaged, child files will not be displayed.

Notes:

None.

Name: Inode Walker

Synopsis

```
#include <user.h>
#include <types.h>
#include <syscall.h>
int inodeTBWalker()
```

Description

Walks through each inode in the inode table using the size specified in the file system's superblock. Get's each inode and populates the corresponding element of a return integer array with 1 if the inode is allocated and 0 if it is not. It then prints the inode number and it's allocation status. Once all inodes have their corresponding array element set to reflect their allocation status, the syscall returns

Return Value

Syscall returns 1.

Errors

None

Notes

None

Name: Compare Walker

Synopsis

```
#include <user.h>
#include <types.h>
#include <syscall.h>
int coWalker()
```

Description

Compares the allocation status of the file system hierarchy and the inode table. Does this by checking the allocation status recorded by the inodeWalker and the directoryWalker in allocation arrays. Compares the allocation status of every inode between the two arrays, and if there are any discrepencies it prints the inode information to the screen. If there are no discrepancies, nothing is printed to the screen. It records inodes with inconsistencies in an integer array. It then returns.

Return Value

Returns 1 if compare completes

Returns -1 if either allocation arrays are uninitialized.

Errors

Error if either inodeWalker or directoryWalker are not called before being used.

Notes

Both inodeWalker and directoryWalker must be called before this syscall. directorWalker must be called from the root directory, without arguments.

Name: Damage Inode

Synopsis

```
#include <user.h>
#include <types.h>
#include <syscall.h>
int damageInode()
```

Description

Damages the allocation value of an inode. Resets the two allocation arrays used by the walker syscalls.

Return Value

Returns -1 if the user tries to damage the root directory.
Returns -1 if the user tries to damage a file that is not a directory
Returns 1 if successful

Errors

Doesn't allow the root directory to be damaged.
Only allows directories to be damaged.

Notes

Must run inodeWalker, directoryWalker, and compareWalker before utilizing this syscall.

Name: Recover Inodes

Synopsis

```
#include <user.h>
#include <types.h>
#include <syscall.h>
int recoverInodes()
```

Description

Uses the results of the compareWalker syscall's inconsistency array to rebuild the file system. Iterates through the array, and if an element is 1 then it updates the corresponding inode. Creates a new inode with a default file name, and re links it to its parent directory.

Return Value

Returns 1 upon completion

Errors

Only errors will be caught by the prior running of directoryWalker, inodeWalker, and compareWalker

Notes

Must run directoryWalker, inodeWalker, and compareWalker after an inode has been damaged in order to populate the inconsistency array this syscall uses. Only then can you utilize this syscall.

Code Changed

```c
int inodeTBWalker(void) {
   for (int i = 0; i < 200; i++)
      inodes[i] = 0;

   struct buf *bp;
   struct dinode *dip;

   cprintf("Allocated Inodes: \n");
   for (int inum = 1; inum < sb.ninodes; inum++) {
      bp = bread(T_DEV, IBLOCK(inum, sb));
      dip = (struct dinode *) bp->data + inum % IPB;
      if (dip->type != 0 && dip->nlink > 0) {
         cprintf("(Indoe %d): 1\n", inum);
         inodes[inum] = 1;
      }
      brelse(bp);
   }

   return 1;
}

// pretty name formatter
char *formatName(char *path) {
   static char buf[DIRSIZ + 1];
   char *p;

   for (p = path + strlen(path); p >= path && *p != '/'; p--);
   p++;

   if (strlen(p) >= DIRSIZ) return p;
   memmove(buf, p, strlen(p));
   memset(buf + strlen(p), ' ', DIRSIZ - strlen(p));
   return buf;
}

// pretty indenter
void printIndent(int indent) {
   for (int i = 0; i < indent; i++)
      cprintf("|  ");
}

static int indent = -1;

int directoryWalker(char *path) {
   struct inode *dp = namei(path);
   if (dp == 0) return -1;

   struct dirent dirEnt;
   ilock(dp);
   indent++;

   if (dp->type == T_DIR) {
      for (uint off = 0; off < dp->size; off += sizeof(dirEnt)) {

         if (readi(dp, (char *) &dirEnt, off, sizeof(dirEnt)) != sizeof(dirEnt));

         if ((strncmp(dirEnt.name, ".", 14) == 0) || (strncmp(dirEnt.name, "..", 14) == 0)) {

            directories[dirEnt.inum] = 1;
            printIndent(indent);
            cprintf("%s INODE: %d\n", formatName(dirEnt.name), dirEnt.inum);

            continue;
         }

         if (dirEnt.inum > 0) {
            struct inode *st = dirlookup(dp, dirEnt.name, 0);
            ilock(st);

            switch (st->type) {
               case T_DIR:
                  iunlock(st);

                  directories[dirEnt.inum] = 1;
                  printIndent(indent);
                  cprintf("%s INODE: %d\n", formatName(dirEnt.name), dirEnt.inum);

                  iunlock(dp);
                  directoryWalker(dirEnt.name);
                  ilock(dp);
                  break;
               case T_FILE:
                  iunlock(st);

                  directories[dirEnt.inum] = 1;
                  printIndent(indent);
                  cprintf("%s INODE: %d\n", formatName(dirEnt.name), dirEnt.inum);

                  break;
               case T_DEV:
                  iunlock(st);
                  directories[dirEnt.inum] = 1;
                  break;
            }
         }

      }
   }
   indent--;
   iunlock(dp);
   return 0;
}

int damageInode(int inum) {

   if (inum <= 1) {
      cprintf("Error: Root\n");
      return -1;
   }

   begin_op();
   struct inode *delNode = iget(T_DIR, inum);

   if (delNode->type != T_DIR) {
      cprintf("Error: Invalid Directory\n");
      return -1;
   }

   // set locks
   ilock(delNode);
   itrunc(delNode);
   iunlockput(delNode);
   end_op();

   cprintf("Damaged Node\n");

   for (int i = 0; i < 100; i++)
      directories[i] = 0;

   return inum;
}

// helper to check directory array
int checkDir(void) {
   for (int i = 0; i < 200; i++)
      if (directories[i] == 1) return 1;
   return -1;
}

// helper to check inode array
int checkInode(void) {
   for (int i = 0; i < 200; i++)
      if (inodes[i] == 1) return 1;
   return -1;
}

int compareWalker(void) {
   if ((checkDir() == -1) || (checkInode() == -1))
      return -1;
   for (int i = 1; i < 200; i++) {
      // indicate missing object
      if ((inodes[i] == 1 && directories[i] == 0) || (inodes[i] == 0 && directories[i] == 1))
         cprintf("Inode: %d missing in one walker\n", i);
      // update missing files for recovery
      compare[i] = inodes[i] ^ directories[i];
   }
   return 1;
}

int recoverWalker(struct inode *recovery_dir) {
   struct inode *dp = iget(T_DIR, 1);
   char fileName[100] = "Recovered Folder";
   for (int i = 1; i < 200; i++)
      if (compare[i] == 1) {
         begin_op();
         dirlink(dp, fileName, i);
         cprintf("Recovered Inode: %d \n", i);
         end_op();
      }
   return 1;
}
```