CS450 Operating Systems (Fall 21)

Programming Assignment 4

File System

Hassan Alamri & Timur Gaimakov

halamri@hawk.iit.edu tgaimakov@hawk.iit.edu

A20473047 A20415319

Designing the new System calls

In this document we will describe the designing of the new system calls according to the file system programming assignment that when a user program calls them, they will act as a set of tools that are used to recover file systems if one or more of their directories inode are damaged. In other words, developing tools that can print files and directories names, print all the allocated inodes, compare inodes and help to recover a damaged file system. We will see the complete explanation and implement of these tools.

The resource files and what we have done to each file:

File 1: usys.S

Going to the **usys.S** file to add the new system calls:

```
SYSCALL(directoryWalker)
SYSCALL(inodeTBWalker)
SYSCALL(compareWalker)
SYSCALL(damageDirInode)
SYSCALL(recoverInode)
```

File 2: syscall.h

Going to the **syscall.h** file to define the new system calls and give them unique numbers, to be recognized when they are called:

```
#define SYS_directoryWalker     23
#define SYS_inodeTBWalker       24
#define SYS_compareWalker       25
#define SYS_damageDirInode      26
#define SYS_recoverInode        27
```

File 3: syscall.c

Adding the new system calls in the ***syscalls[]** array and adding their functions that returns their values so that they can be recognized from their system call numbers.

```
extern int sys_directoryWalker(void);
extern int sys_inodeTBWalker(void);
extern int sys_compareWalker(void);
extern int sys_damageDirInode(void);
extern int sys_recoverInode(void);
static int (*syscalls[])(void) = { …
[SYS_directoryWalker]   sys_directoryWalker,
[SYS_inodeTBWalker]     sys_inodeTBWalker,
[SYS_compareWalker]     sys_compareWalker,
[SYS_damageDirInode]  sys_damageDirInode,
[SYS_recoverInode]      sys_recoverInode, …};
```

File 4: user.h

Going to this file to add the functions and prototypes:

```
int directoryWalker(char*);
int inodeTBWalker(void);
int compareWalker(void);
int damageDirInode(int);
int recoverInode(void);
```

File 5: defs.h

Going to the **defs.h** file to add the functions and prototypes:

```
int directoryWalker(char *);
int inodeTBWalker(void);
int checkDirArray(void);
int checkInodeArray(void);
int compareWalker(void);
int damageInode(int inum);
int recoverWalker();
```

File 6: sysfile.c

Going to the ***sysfile.c*** file to define the functions and their return values:

```
int sys_inodeTBWalker(void) {
        inodeTBWalker();
        return 1;
}

int sys_directoryWalker(void) {
        char *path;
        if (argstr(0, &path) < 0)
                return -1;
        return directoryWalker(path);
}
```

```
int sys_damageDirInode(void) {
        int inum;
        if (argint(0, &inum) < 0)
                return -1;
        return damageInode(inum);
}
int sys_compareWalker(void) {
        return compareWalker();
}

int sys_recoverInode(void) {
        return recoverWalker();
}
```

File 7: fs.c

In this file we add the programs or “the main assignment functions” that support our system calls without changing the original functions or the code. So, we have implemented:

```
int directoryWalker(char *path),
int inodeTBWalker(void),
int compareWalker(void),
int damageInode(int inum),
and int recoverWalker(struct inode *recovery_dir).
```

Moreover, we have implemented other functions that concerned about refinement the outputs which are:
formatName(char *path) that makes printing names look nicer,
printIndent(int indent) that makes printing the file hierarchy look nicer,
checkDir() which checks the directoryWalker allocation array if it has been populated,
and checkInode()that checks the inodeWalker allocation array if it has been populated.

File 7: Makefile

Now before we get into the xv6 OS environment, we need to add the five programs that implement the functions that we already defined. So, we created five driver programs which are:
dirWalker.c: Runs the directoryWalker syscall
inTBWalker.c: Runs the inodeTBWalker syscall
comWalker.c: Runs the compareWalker syscall
damInode.c: Runs damageInode syscall
recInode.c: Runs the recoverInode syscall

The design and why it works for the new system calls

So, the design that we use, is the one that detects and traces the allocation and consistency across walkers using integer arrays that are populated to represent the inode table size. thus, each element in these arrays has its own index which matches an inode. Therefore, we can say that a single element is allocated if its value is 1. Since both the inode and the directory walkers populate their allocation arrays, we can compare their values and store the result into a new array; and thus, the array element is set to 1 if there is any inconsistency. Otherwise, the element is set to 0 which indicates that their allocation elements are equal. The last part is the recovery walker, which uses any consistency array populated by the comparison, and checks what are inodes have been damaged to relink them to their parent directory.

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

Manual Pages

File Name: Directory Walker
Abstract code:

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

Function's description:

The directoryWalker() syscall prints all files and directories names in the file system tree given a specific starting point in the tree. Moreover, it prints inode number beside each name of all files. Populates the corresponding elements of an integer array to reflect the allocation status of each file inode. After printing all files and child files.

Return value:

returns -1, If path is invalid.
returns 0, if path is valid.

Exceptions:

If specified path is invalid, the directoryWalker() would fail.

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

File Name: Inode TB Walker

Abstract code:

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

Function's Description:

Walks through each inode in the inode table using the size specified in the file system's superblock. Gets 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 its allocation status. Once all inodes have their corresponding array element set to reflect their allocation status, the syscall returns

Return Value:

If there is no error, returns 1.

File Name: Compare Walker

Abstract code:

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

Function's 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.

Exceptions:

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

Notes:

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

File Name: Damage Inode

Abstract code:

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

Function's 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

Exceptions:

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.

File Name: Recover Inodes

Abstract code:

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

Function's 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

Exceptions:

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

Notes:

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