

Assignment 4

directoryWalker

Solution Design

1. Create a system call named `fileDirWalker` in `fs.c`
2. Get inode struct from a path of directory.
3. Lock this inode by calling `ilock`
4. If the inode type is `T_FILE` or `T_DEV`, just print relevant info from inode.
5. If the inode type is `T_DIR`:
 - a. print the info from inode at first.
 - b. Because directory is a file containing a sequence dirent structure, we need to visit each of them and calculate the offsets. Once we get the file/directory name from a dirent structure, just concat it to the path to get a new path for the file.
 - c. Recursively call `fileDirWalker` by passing the new path
6. Unlock this inode.
7. Create a user entry point in terms of `directoryWalker` in order to accept user input in shell and call the system call, `fileDirWalker`.

Related files

- `fs.c` - add `fileDirWalker`
- `defs.h` - add function declaration of `fileDirWalker`
- `syscall.c` - add `sys_fileDirWalker`
- `syscall.h` - add system call number named `SYS_fileDirWalker`
- `usys.S` - add `SYSCALL(fileDirWalker)`
- `user.h` - add system call declaration of `fileDirWalker`
- `sysfile.c` - add the implementation of system call named `sys_fileDirWalker` which calls `fileDirWalker`
- `directoryWalker.c` - implement the user entry point for `directoryWalker`

Test Case

From the following screenshot, the user program 'directoryWalker' not only correctly outputs names of files and directories from a given tree, but also prints out inode related info, such as device number, inode number, size and type.

```

$ directoryWalker /
|dir:/ dev:1 inum:1 type:1 size:1536
|file:/README dev:1 inum:2 type:2 size:2286
|file:/cat dev:1 inum:3 type:2 size:14468
|file:/echo dev:1 inum:4 type:2 size:13480
|file:/forktest dev:1 inum:5 type:2 size:8916
|file:/grep dev:1 inum:6 type:2 size:16340
|file:/init dev:1 inum:7 type:2 size:14060
|file:/kill dev:1 inum:8 type:2 size:13524
|file:/ln dev:1 inum:9 type:2 size:13420
|file:/ls dev:1 inum:10 type:2 size:15616
|file:/mkdir dev:1 inum:11 type:2 size:13612
|file:/rm dev:1 inum:12 type:2 size:13588
|file:/sh dev:1 inum:13 type:2 size:24068
|file:/stressfs dev:1 inum:14 type:2 size:14260
|file:/usertests dev:1 inum:15 type:2 size:57184
|file:/wc dev:1 inum:16 type:2 size:15000
|file:/zombie dev:1 inum:17 type:2 size:13252
|file:/directoryWalker dev:1 inum:18 type:2 size:13420
|file:/inodeTBWalker dev:1 inum:19 type:2 size:13320
|file:/rmi dev:1 inum:20 type:2 size:14152
|file:/rvi dev:1 inum:21 type:2 size:14900
|dev:/console dev:1 inum:22 type:3 size:0
$

```

inodeTBWalker

Solution Design

1. Create a system call named fileiTBWalker in fs.c
2. Walk through all the inodes from superblock
 - a. Get a locked buf with the contents of the indicated block
 - b. Get dinode struct from buffer struct
 - c. If it's not a free inode, print out the inode number
 - d. Release the locked buffer by calling brelse
3. Create a user program named inodeTBWalker in order to run it in shell

Related files

- fs.c - add fileiTBWalker
- defs.h - add function declaration of fileiTBWalker for fs.c
- syscall.c - add sys_fileiTBWalker
- syscall.h - add system call number named SYS_fileiTBWalker
- usys.S - add SYSCALL(fileiTBWalker)
- user.h - add system call declaration of fileiTBWalker
- sysfile.c - add the implementation of system call named sys_fileiTBWalker which directly calls fileiTBWalker
- inodeTBWalker.c - implement the user program in order to run it in shell

Test Case

Case 1

The following test case prints out all allocated inodes.

```

$ inodeTBWalker /
inode 1 is used
inode 2 is used
inode 3 is used
inode 4 is used
inode 5 is used
inode 6 is used
inode 7 is used
inode 8 is used
inode 9 is used
inode 10 is used
inode 11 is used
inode 12 is used
inode 13 is used
inode 14 is used
inode 15 is used
inode 16 is used
inode 17 is used
inode 18 is used
inode 19 is used
inode 20 is used
inode 21 is used
inode 22 is used
$ █

```

Case 2

After creating a new file, the inodeTBWalker shows that the inode 23 is used, which is actually used for the new file. So the implementation is in accord with the requirements and expectations.

```

$ cat 'hello' > newfile
$ inodeTBWalker
inode 1 is used
inode 2 is used
inode 3 is used
inode 4 is used
inode 5 is used
inode 6 is used
inode 7 is used
inode 8 is used
inode 9 is used
inode 10 is used
inode 11 is used
inode 12 is used
inode 13 is used
inode 14 is used
inode 15 is used
inode 16 is used
inode 17 is used
inode 18 is used
inode 19 is used
inode 20 is used
inode 21 is used
inode 22 is used
inode 23 is used

```

Comparison of two Walkers

The highlight is path → inode number → inode size; then inode number → buf → dinode → block size. The two values of size should be identical.

Solution Design

1. Create a system call named compareWalker in fs.c
2. Get inode struct from a path of directory.
3. Lock this inode by calling ilock
4. Get a locked buf with the contents of the indicated block which is indexed by the inode number from the inode struct
5. Get dinode struct from buffer struct and get the block size.

6. If the inode type is T_FILE or T_DEV, just print relevant info from inode.
7. If the inode type is T_DIR:
 - a. print the info from inode at first.
 - b. Because directory is a file containing a sequence dirent structure, we need to visit each of them and calculate the offsets. Once we get the file/directory name from a dirent structure, just concat it to the path to get a new path for the file.
 - c. Recursively call compareWalker by passing the new path
8. Unlock this inode.
9. Create a user entry point in terms of compareWalker in order to accept user input in shell and call the system call, compareWalker.

Related files

- fs.c - add compareWalker
- defs.h - add function declaration of compareWalker for fs.c
- syscall.c - add sys_ compareWalker
- syscall.h - add system call number named SYS_ compareWalker
- usys.S - add SYSCALL(compareWalker)
- user.h - add system call declaration of compareWalker
- sysfile.c - add the implementation of system call named sys_ compareWalker which directly calls compareWalker
- compareWalker.c - implement the user program in order to run it in shell

Test Case

From the following plot, we can see that dirWalker is able to generate inode number and data size from inode struct; Then TBWalker uses the previous inode number to get data size from dinode struct. We can easily see the data size from two Walker are identical, which proves they comes from the same inode and the correctness of the two Walkers.

```
$ compareWalker /
[dirWalker] => dir:/ inode:1 size:1536 || [TBWalker-inode:1] => size:1536
[dirWalker] => file:/README inode:2 size:2286 || [TBWalker-inode:2] => size:2286
[dirWalker] => file:/cat inode:3 size:14512 || [TBWalker-inode:3] => size:14512
[dirWalker] => file:/echo inode:4 size:13524 || [TBWalker-inode:4] => size:13524
[dirWalker] => file:/forktest inode:5 size:8956 || [TBWalker-inode:5] => size:8956
[dirWalker] => file:/grep inode:6 size:16384 || [TBWalker-inode:6] => size:16384
[dirWalker] => file:/init inode:7 size:14104 || [TBWalker-inode:7] => size:14104
[dirWalker] => file:/kill inode:8 size:13568 || [TBWalker-inode:8] => size:13568
[dirWalker] => file:/ln inode:9 size:13472 || [TBWalker-inode:9] => size:13472
[dirWalker] => file:/ls inode:10 size:15664 || [TBWalker-inode:10] => size:15664
[dirWalker] => file:/mkdir inode:11 size:13660 || [TBWalker-inode:11] => size:13660
[dirWalker] => file:/rm inode:12 size:13640 || [TBWalker-inode:12] => size:13640
[dirWalker] => file:/sh inode:13 size:24116 || [TBWalker-inode:13] => size:24116
[dirWalker] => file:/stressfs inode:14 size:14304 || [TBWalker-inode:14] => size:14304
[dirWalker] => file:/usertests inode:15 size:57232 || [TBWalker-inode:15] => size:57232
[dirWalker] => file:/wc inode:16 size:15052 || [TBWalker-inode:16] => size:15052
[dirWalker] => file:/zombie inode:17 size:13304 || [TBWalker-inode:17] => size:13304
[dirWalker] => file:/directoryWalker inode:18 size:13464 || [TBWalker-inode:18] => size:13464
[dirWalker] => file:/inodeTBWalker inode:19 size:13368 || [TBWalker-inode:19] => size:13368
[dirWalker] => file:/rmi inode:20 size:14200 || [TBWalker-inode:20] => size:14200
[dirWalker] => file:/rvli inode:21 size:14944 || [TBWalker-inode:21] => size:14944
[dirWalker] => file:/compareWalker inode:22 size:13460 || [TBWalker-inode:22] => size:13460
[dirWalker] => dev:/console inode:23 size:0 || [TBWalker-inode:23] => size:0
$
```

Erase directory inode info

Solution Design

1. Create user program named rmi which erases the information in a directory inode.

2. Walk through all the files coming from user input:
 - a. Get all the data addresses of a file's inode
 - b. Declare and initialize a string for the saving path
 - c. Set the saving path as '/recycle/' plus the filename for future file recovery
 - d. Open the file in 'recycle' and copy those block addresses to the file.
 - e. Unlink the file to remove it from disk.

Related files

- defs.h - add function declaration for getinode
- string.c - add helper function named strcat for string concat operation
- syscall.c - add sys_getinode
- syscall.h - add system call number named SYS_getinode
- usys.S - add SYSCALL(getinode)
- user.h - add system call declaration of getinode
- sysfile.c - add the implementation of system call named sys_getinode which directly calls getinode
- rmi.c - user program for erasing the information in a directory inode

Test Case

Case 1 for erasing a file

First of all, create a new file named newfile containing a simple string.

```
$ echo hello > newfile
$ cat newfile
hello
```

After calling the user program named rmi, we can easily see that the file doesn't exist in the file system by calling ls command. directoryWalker also confirms the inode is gone.

```
$ directoryWalker /
ldir:/ dev:1 inum:1 type:1 size:1728
lfile:/README dev:1 inum:2 type:2 size:2286
lfile:/cat dev:1 inum:3 type:2 size:14512
lfile:/echo dev:1 inum:4 type:2 size:13524
lfile:/forktest dev:1 inum:5 type:2 size:8956
lfile:/grep dev:1 inum:6 type:2 size:16384
lfile:/init dev:1 inum:7 type:2 size:14104
lfile:/kill dev:1 inum:8 type:2 size:13568
lfile:/ln dev:1 inum:9 type:2 size:13472
lfile:/ls dev:1 inum:10 type:2 size:15664
lfile:/mkdir dev:1 inum:11 type:2 size:13660
lfile:/rm dev:1 inum:12 type:2 size:13640
lfile:/sh dev:1 inum:13 type:2 size:24116
lfile:/stressfs dev:1 inum:14 type:2 size:14304
lfile:/usertests dev:1 inum:15 type:2 size:57232
lfile:/wc dev:1 inum:16 type:2 size:15052
lfile:/zombie dev:1 inum:17 type:2 size:13304
lfile:/directoryWalker dev:1 inum:18 type:2 size:13464
lfile:/inodeWalker dev:1 inum:19 type:2 size:13368
lfile:/rmi dev:1 inum:20 type:2 size:14200
lfile:/rvi dev:1 inum:21 type:2 size:14944
lfile:/compareWalker dev:1 inum:22 type:2 size:13460
ldev:/console dev:1 inum:23 type:3 size:0
ldir:/recycle dev:1 inum:24 type:1 size:192
lfile:/recycle/newfile dev:1 inum:27 type:2 size:56
ldir:/recover dev:1 inum:25 type:1 size:128
```

Case 2 for erasing a directory

First of all, create an empty directory named newdir.

Then call directoryWalker to ensure the inode of the created dir exists.

```

$ mkdir newdir
$ directoryWalker /
ldir:/ dev:1 inum:1 type:1 size:1600
lfile:/README dev:1 inum:2 type:2 size:2286
lfile:/cat dev:1 inum:3 type:2 size:14512
lfile:/echo dev:1 inum:4 type:2 size:13524
lfile:/forktest dev:1 inum:5 type:2 size:8956
lfile:/grep dev:1 inum:6 type:2 size:16384
lfile:/init dev:1 inum:7 type:2 size:14104
lfile:/kill dev:1 inum:8 type:2 size:13568
lfile:/ln dev:1 inum:9 type:2 size:13472
lfile:/ls dev:1 inum:10 type:2 size:15664
lfile:/mkdir dev:1 inum:11 type:2 size:13660
lfile:/rm dev:1 inum:12 type:2 size:13640
lfile:/sh dev:1 inum:13 type:2 size:24116
lfile:/stressfs dev:1 inum:14 type:2 size:14304
lfile:/usertests dev:1 inum:15 type:2 size:57232
lfile:/wc dev:1 inum:16 type:2 size:15052
lfile:/zombie dev:1 inum:17 type:2 size:13304
lfile:/directoryWalker dev:1 inum:18 type:2 size:13464
lfile:/inodeT8Walker dev:1 inum:19 type:2 size:13368
lfile:/rmi dev:1 inum:20 type:2 size:14200
lfile:/rvi dev:1 inum:21 type:2 size:14944
lfile:/compareWalker dev:1 inum:22 type:2 size:13460
ldev:/console dev:1 inum:23 type:3 size:0
ldir:/newdir dev:1 inum:24 type:1 size:128

```

After calling the user program named `rmi`, we can easily see that the directory doesn't exist in the file system by calling `ls` command. `directoryWalker` also confirms the inode is gone.

```

$ rmi newfile
$ ls newfile
ls: cannot open newfile
$ directoryWalker /
ldir:/ dev:1 inum:1 type:1 size:1728
lfile:/README dev:1 inum:2 type:2 size:2286
lfile:/cat dev:1 inum:3 type:2 size:14512
lfile:/echo dev:1 inum:4 type:2 size:13524
lfile:/forktest dev:1 inum:5 type:2 size:8956
lfile:/grep dev:1 inum:6 type:2 size:16384
lfile:/init dev:1 inum:7 type:2 size:14104
lfile:/kill dev:1 inum:8 type:2 size:13568
lfile:/ln dev:1 inum:9 type:2 size:13472
lfile:/ls dev:1 inum:10 type:2 size:15664
lfile:/mkdir dev:1 inum:11 type:2 size:13660
lfile:/rm dev:1 inum:12 type:2 size:13640
lfile:/sh dev:1 inum:13 type:2 size:24116
lfile:/stressfs dev:1 inum:14 type:2 size:14304
lfile:/usertests dev:1 inum:15 type:2 size:57232
lfile:/wc dev:1 inum:16 type:2 size:15052
lfile:/zombie dev:1 inum:17 type:2 size:13304
lfile:/directoryWalker dev:1 inum:18 type:2 size:13464
lfile:/inodeT8Walker dev:1 inum:19 type:2 size:13368
lfile:/rmi dev:1 inum:20 type:2 size:14200
lfile:/rvi dev:1 inum:21 type:2 size:14944
lfile:/compareWalker dev:1 inum:22 type:2 size:13460
ldev:/console dev:1 inum:23 type:3 size:0
ldir:/recycle dev:1 inum:24 type:1 size:192
lfile:/recycle/newfile dev:1 inum:27 type:2 size:56
ldir:/recover dev:1 inum:25 type:1 size:128

```

File System Recovery

Solution Design

1. Accept user inputs for files to be recovered.
2. Loop over all files:
 - a. Open the file under “/recycle/” given a filename.
 - b. Read the content of the file including block addresses
 - c. Create another file under “/recover/” with the same filename
 - d. Initialize the file and try to load the data from those block addresses to the buffer
 - e. Write the buffer to the file.
 - f. Close the file

Related files

- `defs.h` – add function declaration for `recoverb`
- `syscall.c` – add `sys_recoverb`
- `syscall.h` – add system call number named `SYS_recoverb`
- `usys.S` – add `SYSCALL(recoverb)`
- `user.h` – add system call declaration of `recoverb`

- sysfile.c – add the implementation of system call named sys_recoverb which directly calls recoverb
- rvi.c – user program for recovering files

Test Case

Case 1 for recovering a file

First of all, create a new file with a string “hello” as its content. Use directoryWalker to find out the inode info.

```
$ echo hello > newfile
$ ./directoryWalker /
|dir:/ dev:1 inum:1 type:1 size:1728
|file:/README dev:1 inum:2 type:2 size:2286
|file:/cat dev:1 inum:3 type:2 size:14512
|file:/echo dev:1 inum:4 type:2 size:13524
|file:/forktest dev:1 inum:5 type:2 size:8956
|file:/grep dev:1 inum:6 type:2 size:16384
|file:/init dev:1 inum:7 type:2 size:14104
|file:/kill dev:1 inum:8 type:2 size:13568
|file:/ln dev:1 inum:9 type:2 size:13472
|file:/ls dev:1 inum:10 type:2 size:15664
|file:/mkdir dev:1 inum:11 type:2 size:13660
|file:/rm dev:1 inum:12 type:2 size:13640
|file:/sh dev:1 inum:13 type:2 size:24116
|file:/stressfs dev:1 inum:14 type:2 size:14304
|file:/usertests dev:1 inum:15 type:2 size:57232
|file:/wc dev:1 inum:16 type:2 size:15052
|file:/zombie dev:1 inum:17 type:2 size:13304
|file:/directoryWalker dev:1 inum:18 type:2 size:13464
|file:/inodeTBWalker dev:1 inum:19 type:2 size:13368
|file:/rmi dev:1 inum:20 type:2 size:14200
|file:/rvi dev:1 inum:21 type:2 size:14944
|file:/compareWalker dev:1 inum:22 type:2 size:13460
|dev:/console dev:1 inum:23 type:3 size:0
|dir:/recycle dev:1 inum:24 type:1 size:192
|dir:/recover dev:1 inum:25 type:1 size:192
|file:/newfile dev:1 inum:27 type:2 size:6
```

Call rvi after newfile destroyed by calling rmi. We can regenerate the file content from /recover/newfile. If we call directoryWalker for the path /recover, we can easily find out the recovered file with the same size as the original one.

```
$ rmi newfile
$ rvi newfile
$ cat /recover/newfile
hello
$ ./directoryWalker /recover
|dir:/recover dev:1 inum:25 type:1 size:192
|file:/recover/newfile dev:1 inum:30 type:2 size:6
$
```

Case 2 for recovering a directory

First of all, create a new directory. Use directoryWalker to find out the inode info.

```
$ mkdir newdir
$ ls newdir
. 1 26 128
. 1 1 1728
$ ./directoryWalker /
|dir:/ dev:1 inum:1 type:1 size:1728
|file:/README dev:1 inum:2 type:2 size:2286
|file:/cat dev:1 inum:3 type:2 size:14512
|file:/echo dev:1 inum:4 type:2 size:13524
|file:/forktest dev:1 inum:5 type:2 size:8956
|file:/grep dev:1 inum:6 type:2 size:16384
|file:/init dev:1 inum:7 type:2 size:14104
|file:/kill dev:1 inum:8 type:2 size:13568
|file:/ln dev:1 inum:9 type:2 size:13472
|file:/ls dev:1 inum:10 type:2 size:15664
|file:/mkdir dev:1 inum:11 type:2 size:13660
|file:/rm dev:1 inum:12 type:2 size:13640
|file:/sh dev:1 inum:13 type:2 size:24116
|file:/stressfs dev:1 inum:14 type:2 size:14304
|file:/usertests dev:1 inum:15 type:2 size:57232
|file:/wc dev:1 inum:16 type:2 size:15052
|file:/zombie dev:1 inum:17 type:2 size:13304
|file:/directoryWalker dev:1 inum:18 type:2 size:13464
|file:/inodeTBWalker dev:1 inum:19 type:2 size:13368
|file:/rmi dev:1 inum:20 type:2 size:14200
|file:/rvi dev:1 inum:21 type:2 size:14944
|file:/compareWalker dev:1 inum:22 type:2 size:13460
|dev:/console dev:1 inum:23 type:3 size:0
|dir:/recycle dev:1 inum:24 type:1 size:128
|dir:/recover dev:1 inum:25 type:1 size:128
|dir:/newdir dev:1 inum:26 type:1 size:128
```

Call rvi after newdir destroyed by calling rmi. We can regenerate the directory from /recover/newdir. If we call directoryWalker for the path /recover, we can easily find out the recovered directory with the same size as the original one.

```
$ rmi newdir
$ ls newdir
ls: cannot open newdir
$ rvi newdir
$ directoryWalker /
ldir:/ dev:1 inum:1 type:1 size:1728
lfile:/README dev:1 inum:2 type:2 size:2286
lfile:/cat dev:1 inum:3 type:2 size:14512
lfile:/echo dev:1 inum:4 type:2 size:13524
lfile:/forktest dev:1 inum:5 type:2 size:8956
lfile:/grep dev:1 inum:6 type:2 size:16384
lfile:/init dev:1 inum:7 type:2 size:14104
lfile:/kill dev:1 inum:8 type:2 size:13568
lfile:/ln dev:1 inum:9 type:2 size:13472
lfile:/ls dev:1 inum:10 type:2 size:15664
lfile:/mkdir dev:1 inum:11 type:2 size:13660
lfile:/rm dev:1 inum:12 type:2 size:13640
lfile:/sh dev:1 inum:13 type:2 size:24116
lfile:/stressfs dev:1 inum:14 type:2 size:14304
lfile:/usertests dev:1 inum:15 type:2 size:57232
lfile:/wc dev:1 inum:16 type:2 size:15052
lfile:/zombie dev:1 inum:17 type:2 size:13304
lfile:/directoryWalker dev:1 inum:18 type:2 size:13464
lfile:/inodeTBWalker dev:1 inum:19 type:2 size:13368
lfile:/rmi dev:1 inum:20 type:2 size:14200
lfile:/rvi dev:1 inum:21 type:2 size:14944
lfile:/compareWalker dev:1 inum:22 type:2 size:13460
ldev:/console dev:1 inum:23 type:3 size:0
ldir:/recycle dev:1 inum:24 type:1 size:192
lfile:/recycle/newdir dev:1 inum:27 type:2 size:56
ldir:/recover dev:1 inum:25 type:1 size:192
lfile:/recover/newdir dev:1 inum:28 type:2 size:128
$ ls /recover/newdir
newdir
$
```

2 28 128