mit-pdos / **xv6-public**

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base fork: **mit-pdos/xv6-public** base: **master** head fork: **pzhzqt/xv6-public** compare: **fs**

✓ **Able to merge.** These branches can be automatically merged.

Create pull request Discuss and review the changes in this comparison with others.

7 commits **17** files changed **0** commit comments **1** contributor

Commits on Feb 12, 2018

pzhzqt `Initialize README` `c01c4be`

Commits on Apr 14, 2018

pzhzqt `Prototype of directoryWalker` `f01d2f0`

Commits on Apr 18, 2018

pzhzqt `Two walkers done` `900b200`

Commits on Apr 19, 2018

pzhzqt `added compare` `e81768c`

Commits on Apr 20, 2018

pzhzqt `Done with erase and recovery.` `4b1bb76`

Commits on Apr 22, 2018

pzhzqt `Recovery finished` `65a0ffe`

Commits on Apr 23, 2018

pzhzqt `Bonus finished.` `fdb53f1`

Showing **17 changed files** with **634 additions** and **1 deletion**. Unified Split

16 Makefile

```
174         _usertests\                          174         _usertests\
175         _wc\                                 175         _wc\
176         _zombie\                             176         _zombie\
                                                 177 +       _directoryWalker\
                                                 178 +       _inodeTBWalker\
                                                 179 +       _compare\
                                                 180 +       _erase\
                                                 181 +       _test\
                                                 182 +       _recoverDir\
                                                 183 +       _damagetype\
                                                 184 +       _recoverType\
177                                              185
178 fs.img: mkfs README $(UPROGS)                 186 fs.img: mkfs README $(UPROGS)
179         ./mkfs fs.img README $(UPROGS)        187         ./mkfs fs.img README $(UPROGS)
```

```
244         mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c
    kill.c\
245         ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c
    zombie.c\
246         printf.c umalloc.c\
247         README dot-bochsrc *.pl toc.* runoff runoff1
    runoff.list\
248         .gdbinit.tmpl gdbutil\
249
```

```
252         mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c
    kill.c\
253         ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c
    zombie.c\
254         printf.c umalloc.c\
255 +               directoryWalker.c\
256 +       inodeTBWalker.c\
257 +       compare.c\
258 +       erase.c\
259 +       test.c\
260 +       recoverDir.c\
261 +       damagetype.c\
262 +       recoverType.c\
263         README dot-bochsrc *.pl toc.* runoff runoff1
    runoff.list\
264         .gdbinit.tmpl gdbutil\
265
```

6 ■■■■■ README

```
...  @@ -1,3 +1,9 @@
```

```
1  xv6 is a re-implementation of Dennis Ritchie's and Ken
    Thompson's Unix
2  Version 6 (v6).  xv6 loosely follows the structure and
    style of v6,
3  but is implemented for a modern x86-based multiprocessor
    using ANSI C.
```

```
1  +USAGE
2  +
3  +
4  +Original README below:
5  +=============================================================
    ==========
6  +
7   xv6 is a re-implementation of Dennis Ritchie's and Ken
    Thompson's Unix
8   Version 6 (v6).  xv6 loosely follows the structure and
    style of v6,
9   but is implemented for a modern x86-based multiprocessor
    using ANSI C.
```

114 ■■■■■ compare.c

```
...  @@ -0,0 +1,114 @@
```

```
1  +#include "types.h"
2  +#include "stat.h"
3  +#include "user.h"
4  +#include "fs.h"
5  +#include "param.h"
6  +
7  +char*
8  +fmtname(char *path)
9  +{
10 +  static char buf[DIRSIZ+1];
11 +  char *p;
12 +
13 +  // Find first character after last slash.
14 +  for(p=path+strlen(path); p >= path && *p != '/'; p--)
15 +    ;
16 +  p++;
17 +
18 +  // Return blank-padded name.
19 +  if(strlen(p) >= DIRSIZ)
20 +    return p;
21 +  memmove(buf, p, strlen(p));
22 +  memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
23 +  return buf;
```

```
+}
+
+void
+directoryWalker(char *path, int *root, short *inode)
+{
+  char buf[512], *p;
+  int fd;
+  struct dirent de;
+  struct stat st;
+
+  if((fd = open(path, 0)) < 0){
+    printf(2, "directoryWalker: cannot open %s\n", path);
+    return;
+  }
+
+  if(fstat(fd, &st) < 0){
+    printf(2, "directoryWalker: cannot stat %s\n", path);
+    close(fd);
+    return;
+  }
+
+  switch(st.type){
+  case T_FILE:
+    printf(1, "%s is not a directory.\n", path);
+    break;
+
+  case T_DIR:
+    if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
+      printf(1, "directoryWalker: path too long\n");
+      break;
+    }
+    strcpy(buf, path);
+    p = buf+strlen(buf);
+    *p++ = '/';
+    while(read(fd, &de, sizeof(de)) == sizeof(de)){
+      if(de.inum == 0)
+        continue;
+      memmove(p, de.name, DIRSIZ);
+      p[DIRSIZ] = 0;
+      if(stat(buf, &st) < 0){
+        printf(1, "directoryWalker: cannot stat %s\n", buf);
+        continue;
+      }
+      char* name=fmtname(buf);
+      if((name[0]=='.'&&*root!=1)||(name[0]=='.'&&name[1]=='.')){
+         continue;
+      }
+      if(name[0]=='.'){
+         *root=0;
+      }
+      printf(1, "name: %s, inode: %d\n", name, st.ino);
+      inode[st.ino]=1;
+      if(st.type==T_DIR && name[0]!='.'){
+         directoryWalker(buf,root,inode);
+      }
+    }
+    break;
+  }
+  close(fd);
```

```
+}
+
+int
+main(int argc, char *argv[])
+{
+       if(argc >= 2){
+               printf(2,"too many arguments\n");
+               exit();
+       }
+       int root=1;
+       short inode_dw[NINODE+1]={0};
+       short inode_iw[NINODE+1]={0};
+
+       directoryWalker(".",&root,inode_dw);
+       iwalk(inode_iw);
+
+       short same=1;
+       for (int i=1;i<NINODE;i++){
+               if (inode_dw[i]==1 && inode_iw[i]==0){
+                       printf(1,"directoryWalker finds inode %d, but inodeTBWalker doesn't\n",i);
+                       same=0;
+               }else if (inode_dw[i]==0 && inode_iw[i]==1){
+                       printf(1,"inodeTBWalker finds inode %d, but directoryWalker doesn't\n",i);
+                       same=0;
+               }
+       }
+       if (same==1){
+               printf(1,"Two walkers find same inodes.\n");
+       }
+
+       exit();
+}
```

18 ■■■■■ damagetype.c

```
@@ -0,0 +1,18 @@
+#include "types.h"
+#include "user.h"
+
+int main(int argc, char* argv[]){
+       if (argc<2){
+               printf(2,"need more argument.\n");
+       }else{
+               for(int i=1;i<argc;i++){
+                       char* path=argv[i];
+                       if(dtype(argv[i])<0){
+                               printf(1,"can't damage %s.\n",path);
+                       }else{
+                               printf(1,"%s damaged.\n",path);
+                       }
+               }
+       }
+       exit();
+}
```

5 defs.h

```
52   int             readi(struct inode*, char*, uint, uint);
53   void            stati(struct inode*, struct stat*);
54   int             writei(struct inode*, char*, uint, uint);
55 -
56   // ide.c
57   void            ideinit(void);
58   void            ideintr(void);
```

```
52   int             readi(struct inode*, char*, uint, uint);
53   void            stati(struct inode*, struct stat*);
54   int             writei(struct inode*, char*, uint, uint);
55 +int                          inodeWalk(short*);
56 +int                          dErase(char*);
57 +void                 recoverDir(struct inode*,struct
   inode*,int*,int);
58 +int                          recoverType(void);
59   // ide.c
60   void            ideinit(void);
61   void            ideintr(void);
```

99 directoryWalker.c

```
... @@ -0,0 +1,99 @@
```

```
1  +#include "types.h"
2  +#include "stat.h"
3  +#include "user.h"
4  +#include "fs.h"
5  +#include "param.h"
6  +
7  +char*
8  +fmtname(char *path)
9  +{
10 +  static char buf[DIRSIZ+1];
11 +  char *p;
12 +
13 +  // Find first character after last slash.
14 +  for(p=path+strlen(path); p >= path && *p != '/'; p--)
15 +    ;
16 +  p++;
17 +
18 +  // Return blank-padded name.
19 +  if(strlen(p) >= DIRSIZ)
20 +    return p;
21 +  memmove(buf, p, strlen(p));
22 +  memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
23 +  return buf;
24 +}
25 +
26 +void
27 +directoryWalker(char *path, int *root, short *inode)
28 +{
29 +  char buf[512], *p;
30 +  int fd;
31 +  struct dirent de;
32 +  struct stat st;
33 +
34 +  if((fd = open(path, 0)) < 0){
35 +    printf(2, "directoryWalker: cannot open %s\n", path);
36 +    return;
37 +  }
38 +
39 +  if(fstat(fd, &st) < 0){
40 +    printf(2, "directoryWalker: cannot stat %s\n", path);
41 +    close(fd);
42 +    return;
43 +  }
44 +
```

```
+  switch(st.type){
+  case T_FILE:
+    printf(1, "%s is not a directory.\n", path);
+    break;
+
+  case T_DIR:
+    if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
+      printf(1, "directoryWalker: path too long\n");
+      break;
+    }
+    strcpy(buf, path);
+    p = buf+strlen(buf);
+    *p++ = '/';
+    while(read(fd, &de, sizeof(de)) == sizeof(de)){
+      if(de.inum == 0)
+        continue;
+      memmove(p, de.name, DIRSIZ);
+      p[DIRSIZ] = 0;
+      if(stat(buf, &st) < 0){
+        printf(1, "directoryWalker: cannot stat %s\n", buf);
+        continue;
+      }
+      char* name=fmtname(buf);
+      if((name[0]=='.'&&*root!=1)||(name[0]=='.'&&name[1]=='.')){
+         continue;
+      }
+      if(name[0]=='.'){
+         *root=0;
+      }
+      printf(1, "name: %s, inode: %d\n", name, st.ino);
+      inode[st.ino]=1;
+      if(st.type==T_DIR && name[0]!='.'){
+         directoryWalker(buf,root,inode);
+      }
+    }
+    break;
+  }
+  close(fd);
+}
+
+int
+main(int argc, char *argv[])
+{
+  int root=1;
+  short inode[NINODE+1]={0};
+
+  if(argc < 2){
+         directoryWalker(".",&root,inode);;
+  }else if(argc>2){
+         printf(2,"too many arguments\n");
+  }else{
+         directoryWalker(argv[1],&root,inode);
+  }
+  exit();
+}
```

17 ■■■■■ erase.c

```
... @@ -0,0 +1,17 @@
```

```
+#include "types.h"
+#include "stat.h"
+#include "user.h"
+#include "param.h"
+
+int
+main(int argc, char *argv[])
+{
+       if (argc==1){
+               printf(1,"need more arguments.\n");
+       }else{
+               for(int i=1;i<argc;i++){
+                       dirErase(argv[i]);
+               }
+       }
+       exit();
+}
```

95 ■■■■■ fs.c

```
 {
   return namex(path, 1, name);
 }
+
+//inode table walker
+int
+inodeWalk(short *inode)
+{
+       uint dev=myproc()->cwd->dev;
+       int inum;
+       struct buf *bp;
+       struct dinode *dip;
+
+       for(inum=1; inum<sb.ninodes; inum++){
+               bp = bread(dev, IBLOCK(inum, sb));
+               dip = (struct dinode*)bp->data + inum%IPB;
+               if(dip->type!=0){
+                       cprintf("inode: %d\n",inum);
+                       inode[inum]=1;
+               }
+               brelse(bp);
+       }
+
+       return 0;
+}
+
+//directory eraser
+int
+dErase(char *path)
+{
+       struct inode *ip=namei(path);
+       ilock(ip);
+       if(ip->inum==1){
+               cprintf("Damaging root directory is not allowed.\n");
+               iunlock(ip);
+               return -1;
+       }
+       if(ip->type!=T_DIR){
+               cprintf("%s is not a directory\n",path);
+               iunlock(ip);
```

```
+               return -1;
+       }else{
+               cprintf("Damaging %s\n",path);
+       }
+       itrunc(ip);
+       iunlockput(ip);
+       return 0;
+}
+
+//recover one damaged directory.
+void
+recoverDir(struct inode* dp,struct inode* ip,int *inum,int num_inum){
+       ilock(ip);
+       ilock(dp);
+       dirlink(ip,".",ip->inum);
+       dirlink(ip,"..",dp->inum);
+       for(int i=0;i<num_inum;i++){
+               char name[6]="file";
+               name[4]=(char)(i+49);
+               name[5]=0;
+               dirlink(ip,name,inum[i]);
+       }
+       iunlockput(dp);
+       iunlockput(ip);
+}
+
+//recover damaged type
+int
+recoverType(){
+       struct buf* bp;
+       struct inode* ip;
+       struct dinode* dip;
+       for(int inum=1;inum<sb.ninodes;inum++){
+               bp = bread(ROOTDEV, IBLOCK(inum, sb));
+               dip = (struct dinode*)bp->data + inum%IPB;
+               brelse(bp);
+               if(dip->type!=0 && dip->size>0){
+                       ip=iget(ROOTDEV,inum);
+                       ilock(ip);
+                       _Bool damaged=0;
+                       struct dirent de;
+                       for(int i=0;i<2;i++){
+                               readi(ip, (char*)&de, i*sizeof(de), sizeof(de));
+                               if((de.name[0]!='.'&&ip->type==T_DIR)||(de.name[0]=='.'&&ip->type==T_FILE)){
+                                       damaged=1;
+                                       break;
+                               }
+                       }
+                       if (damaged){
+                               (ip->type==T_DIR)?(ip->type=T_FILE):(ip->type=T_DIR);
+                               iupdate(ip);
+                               cprintf("inum %d recoverd.\n",ip->inum);
+                       }
+                       iunlockput(ip);
+               }
+       }
```

```
765 +       return 0;
766 +}
```

9 ■■■■■ inodeTBWalker.c

```
... @@ -0,0 +1,9 @@
1 +#include "types.h"
2 +#include "user.h"
3 +#include "param.h"
4 +
5 +int main(){
6 +       short inode[NINODE+1]={0};
7 +       iwalk(inode);
8 +       exit();
9 +}
```

133 ■■■■■ recoverDir.c

```
... @@ -0,0 +1,133 @@
1 +#include "types.h"
2 +#include "stat.h"
3 +#include "user.h"
4 +#include "fs.h"
5 +#include "param.h"
6 +
7 +char*
8 +fmtname(char *path)
9 +{
10 +  static char buf[DIRSIZ+1];
11 +  char *p;
12 +
13 +  // Find first character after last slash.
14 +  for(p=path+strlen(path); p >= path && *p != '/'; p--)
15 +    ;
16 +  p++;
17 +
18 +  // Return blank-padded name.
19 +  if(strlen(p) >= DIRSIZ)
20 +    return p;
21 +  memmove(buf, p, strlen(p));
22 +  memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
23 +  return buf;
24 +}
25 +
26 +void
27 +directoryWalker(char *path, int *root, short *inode,char
rpath[])
28 +{
29 +  char buf[512], *p;
30 +  int fd;
31 +  struct dirent de;
32 +  struct stat st;
33 +
34 +  if((fd = open(path, 0)) < 0){
35 +    printf(2, "directoryWalker: cannot open %s\n", path);
36 +    return;
37 +  }
38 +
39 +  if(fstat(fd, &st) < 0){
40 +    printf(2, "directoryWalker: cannot stat %s\n", path);
41 +    close(fd);
```

```
+    return;
+  }
+
+  switch(st.type){
+  case T_FILE:
+    printf(1, "%s is not a directory.\n", path);
+    break;
+
+  case T_DIR:
+    if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
+      printf(1, "directoryWalker: path too long\n");
+      break;
+    }
+    strcpy(buf, path);
+    p = buf+strlen(buf);
+    *p++ = '/';
+    while(read(fd, &de, sizeof(de)) == sizeof(de)){
+      if(de.inum == 0)
+        continue;
+      memmove(p, de.name, DIRSIZ);
+      p[DIRSIZ] = 0;
+      if(stat(buf, &st) < 0){
+        printf(1, "directoryWalker: cannot stat %s\n", buf);
+        continue;
+      }
+      char* name=fmtname(buf);
+      if((name[0]=='.'&&*root!=1)||(name[0]=='.'&&name[1]=='.')){
+         continue;
+      }
+      if(name[0]=='.'){
+         *root=0;
+      }
+      printf(1, "name: %s, inode: %d\n", name, st.ino);
+      inode[st.ino]=1;
+      if(st.size==0 && st.type==T_DIR && name[0]!='.'){
+         strcpy(rpath,buf);
+      }
+      if(st.type==T_DIR && name[0]!='.'){
+         directoryWalker(buf,root,inode,rpath);
+      }
+    }
+    break;
+  }
+  close(fd);
+}
+
+int recovery(void){
+       int root=1;
+       short inode_dw[NINODE+1]={0};
+       short inode_iw[NINODE+1]={0};
+       char path[256]={0};
+       int inums[NINODE]={0};
+       int num_inum=0;
+       directoryWalker(".",&root,inode_dw,path);
+       iwalk(inode_iw);
+
+       short same=1;
+       for (int i=1;i<NINODE;i++){
+               if (inode_dw[i]==1 && inode_iw[i]==0){
```

```
+                       printf(1,"directoryWalker finds inode %d, but inodeTBWalker doesn't\n",i);
+                       same=0;
+               }else if (inode_dw[i]==0 && inode_iw[i]==1){
+                       printf(1,"inodeTBWalker finds inode %d, but directoryWalker doesn't\n",i);
+                       inums[num_inum++]=i;
+                       same=0;
+               }
+       }
+       if (same==1&&path[0]==0){
+               printf(1,"Recovery finished.\n");
+               return 0;
+       }else{
+               printf(1,"Recovering %s.\n",path);
+               if(recDir(path,inums,num_inum)==0){
+                       printf(1,"Recovery successful.\n");
+                       return 1;
+               }else{
+                       printf(2,"Recovery failed.\n");
+                       return 0;
+               }
+       }
+}
+
+int
+main(int argc, char *argv[])
+{
+       if(argc >= 2){
+               printf(2,"too many arguments\n");
+               exit();
+       }
+       while(recovery());
+       exit();
+}
```

15 ■■■■■ recoverType.c

```
...  @@ -0,0 +1,15 @@
+#include "types.h"
+#include "user.h"
+
+int main(int argc,char* argv[]){
+       if(argc>1){
+               printf(2,"too many arguments.\n");
+       }else{
+               if(recType()==0){
+                       printf(1,"Recover finished.\n");
+               }else{
+                       printf(2,"Recover failed.\n");
+               }
+       }
+       exit();
+}
```

10 ■■■■■ syscall.c

```
 extern int sys_wait(void);
 extern int sys_write(void);
 extern int sys_uptime(void);
```

```
106
107 static int (*syscalls[])(void) = {
108 [SYS_fork]    sys_fork,
126 [SYS_link]    sys_link,
127 [SYS_mkdir]   sys_mkdir,
128 [SYS_close]   sys_close,
129 };
130
131 void
```

```
106 +extern int sys_iwalk(void);
107 +extern int sys_dirErase(void);
108 +extern int sys_recDir(void);
109 +extern int sys_dtype(void);
110 +extern int sys_recType(void);
111
112  static int (*syscalls[])(void) = {
113  [SYS_fork]    sys_fork,
131  [SYS_link]    sys_link,
132  [SYS_mkdir]   sys_mkdir,
133  [SYS_close]   sys_close,
134 +[SYS_iwalk]   sys_iwalk,
135 +[SYS_dirErase]sys_dirErase,
136 +[SYS_recDir]  sys_recDir,
137 +[SYS_dtype]   sys_dtype,
138 +[SYS_recType] sys_recType,
139  };
140
141  void
```

5 ■■■■■ syscall.h

```
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
```

```
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23 +#define SYS_iwalk  22
24 +#define SYS_dirErase 23
25 +#define SYS_recDir 24
26 +#define SYS_dtype  25
27 +#define SYS_recType 26
```

75 ■■■■■ sysfile.c

```
443   fd[1] = fd1;
444   return 0;
445 }
```

```
443   fd[1] = fd1;
444   return 0;
445  }
446 +
447 +//inodeTBwalker
448 +int
449 +sys_iwalk(void)
450 +{
451 +       int addr;
452 +       argint(0,&addr);
453 +       return inodeWalk((short *)addr);
454 +}
455 +
456 +//directory Eraser
457 +int
458 +sys_dirErase(void)
459 +{
460 +       char *path;
461 +       if(argstr(0, &path) < 0){
462 +               return -1;
463 +       }
464 +       begin_op();
465 +       int ret=dErase(path);
466 +       end_op();
467 +       return ret;
468 +}
469 +
470 +//directory Recover
```

```
+int
+sys_recDir(void){
+       char* path;//path that are damaged
+       int* inum;//inums that are not accessible
+       int num_inum;//denote size of inum[]
+       if(argstr(0, &path)<0||argint(1,(int *)&inum)<0||argint(2,&num_inum)<0){
+               return -1;
+       }
+       begin_op();
+       struct inode *dp,*ip;
+       char name[DIRSIZ];
+       dp=nameiparent(path,name);
+       ip=namei(path);
+       recoverDir(dp,ip,inum,num_inum);
+       end_op();
+       return 0;
+}
+
+//type damage
+int
+sys_dtype(void){
+       char *path;
+       if(argstr(0,&path)<0){
+               return -1;
+       }
+       struct inode *ip;
+       if((ip=namei(path))==0){
+               return -1;
+       }
+       begin_op();
+       ilock(ip);
+       if(ip->type==T_DIR){
+               ip->type=T_FILE;
+       }else if(ip->type==T_FILE){
+               ip->type=T_DIR;
+       }
+       iupdate(ip);
+       iunlockput(ip);
+       end_op();
+       return 0;
+}
+
+//type recover
+int
+sys_recType(void){
+       begin_op();
+       int ret=recoverType();
+       end_op();
+       return ret;
+}
```

8 ■■■■■ test.c

```
@@ -0,0 +1,8 @@
+//This file is only for testing purpose
+#include "types.h"
+#include "user.h"
+
+int main(){
+       printf(1,"test test\n");
```

```
                                                  +       exit();
                                                  +}
```

5 ■■■■■ user.h

```
 char* sbrk(int);                                 char* sbrk(int);
 int sleep(int);                                  int sleep(int);
 int uptime(void);                                int uptime(void);
                                                  +int iwalk(short*);
                                                  +int dirErase(char*);
                                                  +int recDir(char*,int*,int);
                                                  +int dtype(char*);
                                                  +int recType(void);

 // ulib.c                                        // ulib.c
 int stat(char*, struct stat*);                   int stat(char*, struct stat*);
```

5 ■■■■■ usys.S

```
 SYSCALL(sbrk)                                    SYSCALL(sbrk)
 SYSCALL(sleep)                                   SYSCALL(sleep)
 SYSCALL(uptime)                                  SYSCALL(uptime)
                                                  +SYSCALL(iwalk)
                                                  +SYSCALL(dirErase)
                                                  +SYSCALL(recDir)
                                                  +SYSCALL(dtype)
                                                  +SYSCALL(recType)
```

No commit comments for this range