MARCO ARES & TRAVIS CRIPE

On top of your hard copy: MARK yes or no

1. does your ls work? ls; ls /dir1; ls /dir1/dir3 **YES**

2. does your cd work? cd /dir1; cd /dir1/dir3 **YES**

3. does your pwd work? **YES**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* type.h file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <time.h>

#include <libgen.h>

#include <sys/stat.h>

#include <ext2fs/ext2\_fs.h>

#include <dirent.h>

typedef unsigned char u8;

typedef unsigned short u16;

typedef unsigned int u32;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR;

SUPER \*sp;

GD \*gp;

INODE \*ip;

DIR \*dp;

#define FREE 0

#define READY 1

#define BLKSIZE 1024

#define NMINODE 64

#define NFD 8

#define NPROC 2

typedef struct minode{

INODE INODE;

int dev, ino;

int refCount;

int dirty;

// for level-3

int mounted;

struct mntable \*mptr;

}MINODE;

typedef struct oft{ // for level-2

int mode;

int refCount;

MINODE \*mptr;

int offset;

}OFT;

typedef struct proc{

struct proc \*next;

int pid;

int uid;

int status;

MINODE \*cwd;

OFT \*fd[NFD]; }PROC;

/\*\*\*\*\*\*\*\*\*\*\*\*\* cd\_ls\_pwd.c file \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\* globals defined in main.c file \*\*\*\*/

extern MINODE minode[NMINODE];

extern MINODE \*root;

extern PROC proc[NPROC], \*running;

extern char gpath[256];

extern char \*name[64];

extern int n;

extern int fd, dev;

extern int nblocks, ninodes, bmap, imap, inode\_start;

extern char line[256], cmd[32], pathname[256];

#define OWNER 000700

#define GROUP 000070

#define OTHER 000007

int change\_dir(char \*pathname)

{

int ino = getino(pathname);

if (ino == 0) //return error if ino = 0

return 0;

MINODE \*mip = iget(dev, ino);

if (!S\_ISDIR(mip->INODE.i\_mode)) //return error if mip is not DIR

{

printf("ERROR\n");

iput(mip);

return 0;

}

iput(running->cwd); //release old cwd;

running->cwd = mip; // change cwd to mip

}

int ls(char \*pathname)

{

int ino = getinto(pathname);

MINODE \*mip = iget(dev, ino);

int r;

char \*path;

pathname = "./";

if (argc > 1)

{

pathname = argv[1];

}

if (r = lstat(pathname, mip->INODE->i\_block[0]) < 0)

{

printf("no such file %s \n", pathname);

exit(1);

}

strcpy(path, pathname);

if (path[0] != '/')

{

getcwd(cwd, 256);

strcpy(path, cwd);

strcat(path, "/");

strcat(path, pathname);

if (entry[1].value != NULL)

{

strcat(path, "/");

strcat(path, entry[1].value);

}

}

if (S\_ISDIR(mip->INODE->i\_mode))

{

ls\_dir(path);

}

else

{

ls\_file(path);

}

}

int ls\_file(int ino, char \*name)

{

MINODE \*mip = iget(dev, ino);

int r, i;

char ftime[64];

if ((mip->INODe->i\_mode & 0xF000) == 0x8000) // if (S\_ISREG())

{

printf("%c", '-');

}

else if ((mip->INODe->i\_mode & 0xF000) == 0x4000) // if (S\_ISDIR())

{

printf("%c", 'd');

}

else if ((mip->INODe->i\_mode & 0xF000) == 0xA000) // if (S\_ISLNK())

{

printf("%c", 'l');

}

for (i = 8; i >= 0; i--)

{

if (mip->INODe->i\_mode & (1 << i)) // print r|w|x

{

printf("%c", t1[i]);

}

else

{

printf("%c", t2[i]); // or print -

}

}

char ftime[64];

printf("%4d ", mip->INODE->i\_uid); // uid

printf("%8d ", mip->INODE->st\_size); // size

strcpy(ftime, ctime(&mip->INODE->ctime)); // print time in calendar form

ftime[strlen(ftime) - 1] = 0; // kill \n at end

printf("%s ", ftime); // print name

printf("%s", basename(fname)); // print file basename // print -> linkname if symbolic file

if ((mip->INODe->i\_mode & 0xF000) == 0xA000)

{

char\* linkname = "";

r = readlink(fname, linkname, 16);

printf(" -> %s", linkname);

}

printf("\n");

}

int ls\_dir(char \*dirname)

{

int ino = getinto(dirname);

MINODE \*mip = iget(dev, ino);

char sbuf[BLKSIZE], temp[256];

char \*cp;

int i;

for (i = 0; i < 12; i++) { // assume DIR at most 12 direct blocks

if (mip->INODE.i\_block[i] == 0)

break;

// YOU SHOULD print i\_block[i] number here

get\_block(dev, mip->INODE.i\_block[i], sbuf);

dp = (DIR \*)sbuf;

cp = sbuf;

while (cp < sbuf + BLKSIZE) {

strncpy(temp, dp->name, dp->name\_len);

temp[dp->name\_len] = 0;

printf("%4d %4d %4d %s\n",

dp->inode, dp->rec\_len, dp->name\_len, temp);

ls\_file(dp, temp);

cp += dp->rec\_len;

dp = (DIR \*)cp;

}

}

}

int pwd(MINODE \*wd)

{

if (wd == root)

printf("/\n");

else

rpwd(wd);

}

int rpwd(MINODE \*wd)

{

int my\_ino, parent\_ino;

char myname[256], buf[BLKSIZE], \*cp;

MINODE \*pip;

if (wd == root)

return;

get\_block(dev, wd->INODE.i\_block[0], buf);

dp = (DIR \*)buf;

cp = buf;

my\_ino = dp->inode;

cp += dp->rec\_len;

dp = (DIR \*)cp;

parent\_ino = dp->inode;

pip = iget(dev, parent\_ino);

// using show dir function to get the string name from dir as LOCAL

pwd\_dir(pip, myname, buf, my\_ino);

rpwd(pip);

printf("/%s", myname);

}

int pwd\_dir(MINODE\* ip, char myname[], char sbuf[], int my\_ino)

{

char\* cp;

int i;

for (i = 0; i < 12; i++) { // assume DIR at most 12 direct blocks

if (ip->INODE.i\_block[i] != 0) {

get\_block(dev, ip->INOIDE.i\_block[i], sbuf);

dp = (DIR\*)sbuf;

cp = sbuf;

while (cp < sbuf + BLKSIZE) {

// finidng the current dir

if (dp->inode == my\_ino) {

strncpy(myname, dp->name, dp->name\_len);

myname[dp->name\_len] = 0;

}

cp += dp->rec\_len;

dp = (DIR\*)cp;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\* util.c file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\* globals defined in main.c file \*\*\*\*/

#include "type.h"

extern MINODE minode[NMINODE];

extern MINODE \*root;

extern PROC proc[NPROC], \*running;

extern char gpath[256];

extern char \*name[64];

extern int n;

extern int fd, dev;

extern int nblocks, ninodes, bmap, imap, inode\_start;

extern char line[256], cmd[32], pathname[256];

int get\_block(int dev, int blk, char \*buf)

{

lseek(dev, (long)blk\*BLKSIZE, 0);

read(dev, buf, BLKSIZE);

}

int put\_block(int dev, int blk, char \*buf)

{

lseek(dev, (long)blk\*BLKSIZE, 0);

write(dev, buf, BLKSIZE);

}

int tokenize(char \*pathname)

{

char\* s;

strcpy(gline, pathname);

nname = 0;

s = strtok(gline, "/");

while (s)

{

name[nname++] = s;

s = strtok(0, "/");

}

}

// return minode pointer to loaded INODE

MINODE \*iget(int dev, int ino)

{

int i;

MINODE \*mip;

char buf[BLKSIZE];

int blk, disp;

INODE \*ip;

for (i=0; i<NMINODE; i++){

mip = &minode[i];

if (mip->refCount && mip->dev == dev && mip->ino == ino){

mip->refCount++;

printf("found [%d %d] as minode[%d] in core\n", dev, ino, i);

return mip;

}

}

for (i=0; i<NMINODE; i++){

mip = &minode[i];

if (mip->refCount == 0){

//printf("allocating NEW minode[%d] for [%d %d]\n", i, dev, ino);

mip->refCount = 1;

mip->dev = dev;

mip->ino = ino;

// get INODE of ino to buf

blk = (ino-1) / 8 + inode\_start;

disp = (ino-1) % 8;

//printf("iget: ino=%d blk=%d disp=%d\n", ino, blk, disp);

get\_block(dev, blk, buf);

ip = (INODE \*)buf + disp;

// copy INODE to mp->INODE

mip->INODE = \*ip;

return mip;

}

}

printf("PANIC: no more free minodes\n");

return 0;

}

int iput(MINODE \*mip)

{

int i, block, offset;

char buf[BLKSIZE];

INODE \*ip;

if (mip==0)

return;

mip->refCount--;

if (mip->refCount > 0) return;

if (!mip->dirty) return;

/\* write back \*/

printf("iput: dev=%d ino=%d\n", mip->dev, mip->ino);

block = ((mip->ino - 1) / 8) + inode\_start;

offset = (mip->ino - 1) % 8;

/\* first get the block containing this inode \*/

get\_block(mip->dev, block, buf);

ip = (INODE \*)buf + offset;

\*ip = mip->INODE;

put\_block(mip->dev, block, buf);

}

int search(MINODE \*mip, char \*name)

{

int i;

char\* cp, temp[256], sbuf[BLKSIZE];

for (i = 0; i < 12; i++) // serach DIR direct blocks only

{

if (mip->INODE.i\_block[i] == 0)

return 0;

get\_block(dev, mip->INODE.i\_block[i], sbuf);

dp = (DIR\*)sbuf;

cp = sbuf;

while (cp < sbuf + BLKSIZE)

{

strncpy(temp, dp->name, dp->name\_len);

temp[dp->name\_len] = 0;

if (strcmp(name, temp) == 0)

{

printf("found %s : inumber = %d\n", name, dp->inode);

return dp->inode;

}

cp += dp->rec\_len;

dp = (DIR\*)cp;

}

}

return 0;

}

int getino(char \*pathname)

{

int i, ino, blk, disp;

INODE \*ip;

MINODE \*mip;

printf("getino: pathname=%s\n", pathname);

if (strcmp(pathname, "/")==0)

return 2;

if (pathname[0]=='/')

mip = iget(dev, 2);

else

mip = iget(running->cwd->dev, running->cwd->ino);

tokenize(pathname);

for (i=0; i<n; i++){

printf("===========================================\n");

ino = search(mip, name[i]);

if (ino==0){

iput(mip);

printf("name %s does not exist\n", name[i]);

return 0;

}

iput(mip);

mip = iget(dev, ino);

}

iput(mip);

return ino;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* KCW testing ext2 file system \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <time.h>

#include <libgen.h>

#include <sys/stat.h>

#include <ext2fs/ext2\_fs.h>

#include "type.h"

MINODE minode[NMINODE];

MINODE \*root;

PROC proc[NPROC], \*running;

char gpath[256]; // global for tokenized components

char \*name[64]; // assume at most 64 components in pathname

int n; // number of component strings

int fd, dev;

int nblocks, ninodes, bmap, imap, inode\_start;

char line[256], cmd[32], pathname[256];

#include "util.c"

#include "cd\_ls\_pwd.c"

int init()

{

int i, j;

MINODE \*mip;

PROC \*p;

printf("init()\n");

for (i=0; i<NMINODE; i++){

mip = &minode[i];

mip->dev = mip->ino = 0;

mip->refCount = 0;

mip->mounted = 0;

mip->mptr = 0;

}

for (i=0; i<NPROC; i++){

p = &proc[i];

p->pid = i;

p->uid = i;

p->cwd = 0;

p->status = READY;

for (j=0; j<NFD; j++)

p->fd[j] = 0;

}

proc[NPROC - 1].next = &proc[0];

running = &proc[0];

}

// load root INODE and set root pointer to it

int mount\_root()

{

int i;

MTABLE \*mp;

SUPER \*sp;

GD \*gp;

char buf[BLKSIZE];

dev = open(rootdev, O\_RDWR);

if (dev < 0) {

printf("panic : can’t open root device\n");

exit(1);

}

/\* get super block of rootdev \*/

get\_block(dev, 1, buf);

sp = (SUPER \*)buf;

/\* check magic number \*/

if (sp->s\_magic != SUPER\_MAGIC) {

printf("super magic=%x : %s is not an EXT2 filesys\n",

sp->s\_magic, rootdev);

exit(0);

}

// fill mount table mtable[0] with rootdev information

mp = &mtable[0]; // use mtable[0]

mp->dev = dev;

// copy super block info into mtable[0]

ninodes = mp->ninodes = sp->s\_inodes\_count;

nblocks = mp->nblocks = sp->s\_blocks\_count;

strcpy(mp->devName, rootdev);

strcpy(mp->mntName, "/");

get\_block(dev, 2, buf);

gp = (GD \*)buf;

bmap = mp->bmap = gp->bg\_blocks\_bitmap;

imap = mp->imap = gp->bg\_inodes\_bitmap;

iblock = mp->iblock = gp->bg\_inode\_table;

printf("bmap=%d imap=%d iblock=%d\n", bmap, imap iblock);

// call iget(), which inc minode’s refCount

root = iget(dev, 2); // get root inode

mp->mntDirPtr = root; // double link

root->mntPtr = mp;

// set proc CWDs

for (i = 0; i < NPROC; i++) // set proc’s CWD

proc[i].cwd = iget(dev, 2); // each inc refCount by 1

printf("mount : %s mounted on / \n", rootdev);

return 0;

}

char \*disk = "mydisk";

int main(int argc, char \*argv[ ])

{

int ino;

char buf[BLKSIZE];

if (argc > 1)

disk = argv[1];

printf("checking EXT2 FS ....");

if ((fd = open(disk, O\_RDWR)) < 0){

printf("open %s failed\n", disk); exit(1);

}

dev = fd;

/\*\*\*\*\*\*\*\*\*\* read super block at 1024 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

get\_block(dev, 1, buf);

sp = (SUPER \*)buf;

/\* verify it's an ext2 file system \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (sp->s\_magic != 0xEF53){

printf("magic = %x is not an ext2 filesystem\n", sp->s\_magic);

exit(1);

}

printf("OK\n");

ninodes = sp->s\_inodes\_count;

nblocks = sp->s\_blocks\_count;

get\_block(dev, 2, buf);

gp = (GD \*)buf;

bmap = gp->bg\_block\_bitmap;

imap = gp->bg\_inode\_bitmap;

inode\_start = gp->bg\_inode\_table;

printf("bmp=%d imap=%d inode\_start = %d\n", bmap, imap, inode\_start);

init();

mount\_root();

printf("root refCount = %d\n", root->refCount);

printf("creating P0 as running process\n");

running = &proc[0];

running->status = READY;

running->cwd = iget(dev, 2);

printf("root refCount = %d\n", root->refCount);

//printf("hit a key to continue : "); getchar();

while(1){

printf("input command : [ls|cd|pwd|quit] ");

fgets(line, 128, stdin);

line[strlen(line)-1] = 0;

if (line[0]==0)

continue;

pathname[0] = 0;

cmd[0] = 0;

sscanf(line, "%s %s", cmd, pathname);

printf("cmd=%s pathname=%s\n", cmd, pathname);

if (strcmp(cmd, "ls")==0)

list\_file();

if (strcmp(cmd, "cd")==0)

change\_dir();

if (strcmp(cmd, "pwd")==0)

pwd(running->cwd);

if (strcmp(cmd, "quit")==0)

quit();

}

}

int quit()

{

int i;

MINODE \*mip;

for (i=0; i<NMINODE; i++){

mip = &minode[i];

if (mip->refCount > 0)

iput(mip);

}

exit(0);

}

1. EXPLAIN in words: What does mount root do? **It attaches the file to the directory so that we can access the files**

2. Given int ino; // an INODE number

INODE thisInode; // an INODE structure

Write C code to load the INODE of ino into thisInode

3. Assume: MINODE \*mip points at an in-memory DIR minode

INODE

dev, ino

refCount

dirty