Kyler Little

/\*\*\*\*\*\*\*\*\* super.c code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

#include "constants.h"

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

#define BLKSIZE 1024

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* in <ext2fs/ext2\_fs.h>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct ext2\_super\_block {

u32 s\_inodes\_count; // total number of inodes

u32 s\_blocks\_count; // total number of blocks

u32 s\_r\_blocks\_count;

u32 s\_free\_blocks\_count; // current number of free blocks

u32 s\_free\_inodes\_count; // current number of free inodes

u32 s\_first\_data\_block; // first data block in this group

u32 s\_log\_block\_size; // 0 for 1KB block size

u32 s\_log\_frag\_size;

u32 s\_blocks\_per\_group; // 8192 blocks per group

u32 s\_frags\_per\_group;

u32 s\_inodes\_per\_group;

u32 s\_mtime;

u32 s\_wtime;

u16 s\_mnt\_count; // number of times mounted

u16 s\_max\_mnt\_count; // mount limit

u16 s\_magic; // 0xEF53

// A FEW MORE non-essential fields

};

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

char buf[BLKSIZE];

int fd;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int super()

{

// read SUPER block

get\_block(fd, 1, buf);

sp = (SUPER \*)buf;

// check for EXT2 magic number:

printf("s\_magic = %x\n", sp->s\_magic);

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

printf("NOT an EXT2 FS\n");

exit(1);

}

printf("EXT2 FS OK\n");

printf("s\_inodes\_count = %d\n", sp->s\_inodes\_count);

printf("s\_blocks\_count = %d\n", sp->s\_blocks\_count);

printf("s\_free\_inodes\_count = %d\n", sp->s\_free\_inodes\_count);

printf("s\_free\_blocks\_count = %d\n", sp->s\_free\_blocks\_count);

printf("s\_first\_data\_blcok = %d\n", sp->s\_first\_data\_block);

printf("s\_log\_block\_size = %d\n", sp->s\_log\_block\_size);

printf("s\_blocks\_per\_group = %d\n", sp->s\_blocks\_per\_group);

printf("s\_inodes\_per\_group = %d\n", sp->s\_inodes\_per\_group);

printf("s\_mnt\_count = %d\n", sp->s\_mnt\_count);

printf("s\_max\_mnt\_count = %d\n", sp->s\_max\_mnt\_count);

printf("s\_magic = %x\n", sp->s\_magic);

printf("s\_mtime = %s", ctime(&sp->s\_mtime));

printf("s\_wtime = %s", ctime(&sp->s\_wtime));

}

char \*disk = "../mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

printf("open failed\n");

exit(1);

}

super();

}

**super.C output:**

s\_magic = ef53

EXT2 FS OK

s\_inodes\_count = 184

s\_blocks\_count = 1440

s\_free\_inodes\_count = 155

s\_free\_blocks\_count = 1389

s\_first\_data\_blcok = 1

s\_log\_block\_size = 0

s\_blocks\_per\_group = 8192

s\_inodes\_per\_group = 184

s\_mnt\_count = 1

s\_max\_mnt\_count = -1

s\_magic = ef53

s\_mtime = Wed Oct 24 20:20:58 2018

s\_wtime = Wed Oct 24 20:20:58 2018

/\*\*\*\*\*\*\*\*\* gd.c code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

#include "constants.h"

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

#define BLKSIZE 1024

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* in <ext2fs/ext2\_fs.h>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct ext2\_group\_desc

{

u32 bg\_block\_bitmap; // Blocks bitmap block

u32 bg\_inode\_bitmap; // Inodes bitmap block

u32 bg\_inode\_table; // Inodes table block

u16 bg\_free\_blocks\_count; // Free blocks count

u16 bg\_free\_inodes\_count; // Free inodes count

u16 bg\_used\_dirs\_count; // Directories count

u16 bg\_flags;

u32 bg\_exclude\_bitmap\_lo; // Exclude bitmap for snapshots

u16 bg\_block\_bitmap\_csum\_lo;// crc32c(s\_uuid+grp\_num+bitmap) LSB

u16 bg\_inode\_bitmap\_csum\_lo;// crc32c(s\_uuid+grp\_num+bitmap) LSB

u16 bg\_itable\_unused; // Unused inodes count

u16 bg\_checksum; // crc16(s\_uuid+group\_num+group\_desc)

};

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

char buf[BLKSIZE];

int fd;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int gd()

{

// read GROUP DESCRIPTOR block

get\_block(fd, 2, buf);

gp = (GD \*)buf;

if (!gp) {

printf("No GROUP DESCRIPTOR block!\n");

exit(1);

}

printf("EXT2 FS OK\n");

printf("bg\_block\_bitmap = %d\n", gp->bg\_block\_bitmap);

printf("bg\_inode\_bitmap = %d\n", gp->bg\_inode\_bitmap);

printf("bg\_inode\_table = %d\n", gp->bg\_inode\_table);

printf("bg\_free\_inodes\_count = %d\n", gp->bg\_free\_inodes\_count);

printf("bg\_free\_blocks\_count = %d\n", gp->bg\_free\_blocks\_count);

printf("bg\_used\_dirs\_count = %d\n", gp->bg\_used\_dirs\_count);

printf("bg\_flags = %d\n", gp->bg\_flags);

printf("bg\_exclude\_bitmap\_lo = %d\n", gp->bg\_exclude\_bitmap\_lo);

printf("bg\_inode\_bitmap\_csum\_lo = %d\n", gp->bg\_inode\_bitmap\_csum\_lo);

printf("bg\_block\_bitmap\_csum\_lo = %d\n", gp->bg\_block\_bitmap\_csum\_lo);

printf("bg\_itable\_unused = %d\n", gp->bg\_itable\_unused);

printf("bg\_checksum = %d\n", gp->bg\_checksum);

}

char \*disk = "../mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

printf("open failed\n");

exit(1);

}

gd();

}

**gd.c output:**

EXT2 FS OK

bg\_block\_bitmap = 8

bg\_inode\_bitmap = 9

bg\_inode\_table = 10

bg\_free\_inodes\_count = 155

bg\_free\_blocks\_count = 1389

bg\_used\_dirs\_count = 6

bg\_flags = 4

bg\_exclude\_bitmap\_lo = 0

bg\_inode\_bitmap\_csum\_lo = 0

bg\_block\_bitmap\_csum\_lo = 0

bg\_itable\_unused = 0

bg\_checksum = 0

//imap

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

// define shorter TYPES, save typing efforts

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR; // need this for new version of e2fs

#define BLKSIZE 1024

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

char buf[BLKSIZE];

int fd;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int tst\_bit(char \*buf, int bit)

{

int i, j;

i = bit / 8; j = bit % 8;

if (buf[i] & (1 << j))

return 1;

return 0;

}

int imap()

{

char buf[BLKSIZE];

int imap, ninodes;

int i;

// read SUPER block

get\_block(fd, 1, buf);

sp = (SUPER \*)buf;

ninodes = sp->s\_inodes\_count;

printf("ninodes = %d\n", ninodes);

// read Group Descriptor 0

get\_block(fd, 2, buf);

gp = (GD \*)buf;

imap = gp->bg\_inode\_bitmap;

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

// read inode\_bitmap block

get\_block(fd, imap, buf);

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

(tst\_bit(buf, i)) ? putchar('1') : putchar('0');

if (i && (i % 8)==0)

printf(" ");

}

printf("\n");

}

char \*disk = "mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

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

exit(1);

}

imap();

}

**imap.c output:**

ninodes = 184

imap = 9

111111111 11111111 11111110 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000000

//bmap

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <ext2fs/ext2\_fs.h>

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR;

#define BLKSIZE 1024

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

char buf[BLKSIZE];

int fd;

int get\_block(int fd, int blk, char buf[])

{

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

read(fd, buf, BLKSIZE);

}

int tst\_bit(char \*buf, int bit)

{

int i, j;

i = bit / 8; j = bit % 8;

if (buf[i] & (1 << j))

return 1;

return 0;

}

int bmap()

{

int bmap, i, nblocks;

get\_block(fd,1,buf);

sp=(SUPER \*)buf;

nblocks=sp->s\_blocks\_count;

printf("nblocks = %d\n", nblocks);

get\_block(fd,2,buf);

gp=(GD \*)buf;

bmap=gp->bg\_block\_bitmap;

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

get\_block(fd,bmap,buf);

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

putchar(tst\_bit(buf,i)+48);

if(i && (i%8)==0)

putchar(' ');

if(i && (i%32) ==0)

putchar('\n');

}

putchar('\n');

}

char \*disk = "mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

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

exit(1);

}

bmap();

}

**bmap.c output:**

nblocks = 1440

bmap = 8

111111111 11111111 11111111 11111111

11111111 11111111 11110000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 0001101

/\*\*\*\*\*\*\*\*\* inode.c: print information in / INODE (INODE #2) \*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

#include "constants.h"

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

int fd;

int iblock;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int inode()

{

char buf[BLKSIZE];

// read GD

get\_block(fd, 2, buf);

gp = (GD \*)buf;

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

printf("%8d %8d %8d %8d %8d %8d\n",

gp->bg\_block\_bitmap,

gp->bg\_inode\_bitmap,

gp->bg\_inode\_table,

gp->bg\_free\_blocks\_count,

gp->bg\_free\_inodes\_count,

gp->bg\_used\_dirs\_count);

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

iblock = gp->bg\_inode\_table; // get inode start block#

printf("inode\_block=%d\n", iblock);

// get inode start block

get\_block(fd, iblock, buf);

ip = (INODE \*)buf + 1; // ip points at 2nd INODE

printf("mode=0x%4x\n", ip->i\_mode);

printf("uid=%d gid=%d\n", ip->i\_uid, ip->i\_gid);

printf("size=%d\n", ip->i\_size);

printf("time=%s", ctime(&ip->i\_ctime));

printf("link=%d\n", ip->i\_links\_count);

printf("i\_block[0]=%d\n", ip->i\_block[0]);

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

u16 i\_mode; // same as st\_imode in stat() syscall

u16 i\_uid; // ownerID

u32 i\_size; // file size in bytes

u32 i\_atime; // time fields

u32 i\_ctime;

u32 i\_mtime;

u32 i\_dtime;

u16 i\_gid; // groupID

u16 i\_links\_count; // link count

u32 i\_blocks; // IGNORE

u32 i\_flags; // IGNORE

u32 i\_reserved1; // IGNORE

u32 i\_block[15]; // IMPORTANT, but later

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

}

char \*disk = "../mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

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

exit(1);

}

inode();

}

**inode.c output:**

inode\_block=10

mode=0x41ed

uid=0 gid=0

size=1024

time=Wed Oct 24 20:20:58 2018

link=7

i\_block[0]=33

/\*\*\*\*\*\*\*\*\* dir.c: print all entries under '/' directory \*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

#include <string.h>

#include "constants.h"

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

int fd;

int iblock;

int search(INODE \*ip, char \*name) {

int i = 0;

char dbuf[BLKSIZE], \*cp;

for (i = 0; i < NUM\_DIRECT\_BLKS; i++) {

if (ip->i\_block[i] == 0) return 0; // NOT FOUND

// Otherwise, search for name in dir; read direct block into dbuf

get\_block(fd, ip->i\_block[i], dbuf);

dp = (DIR \*)dbuf;

cp = dbuf;

while (cp < dbuf + BLKSIZE) {

// if name matches with dir's name, return inode number

if (!strncmp(dp->name, name, dp->name\_len)) {

return dp->inode;

}

cp += dp->rec\_len;

dp = (DIR \*)cp;

}

}

}

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int dir()

{

char buf[BLKSIZE], dbuf[BLKSIZE], \*cp, temp[MAX\_FILENAME\_LEN];

// read GD

get\_block(fd, 2, buf);

gp = (GD \*)buf;

iblock = gp->bg\_inode\_table; // get inode start block#

printf("inode\_block = %d\n", iblock);

// get inode start block

get\_block(fd, iblock, buf);

// get root inode #2

ip = (INODE \*)buf + 1; // ip points at 2nd INODE

// int ino = search(ip, "dir2");

// if (ino) printf("ino: %d\n", ino);

// ip = (INODE \*)buf + 1; // ip points at 2nd INODE

int i;

for (i = 0; i<NUM\_DIRECT\_BLKS; i++) {

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

break;

}

// Note: ip->i\_block[0-11] will yield a pointer to a direct block

printf("i\_block[%d] = %d\n", i, ip->i\_block[i]);

// Read direct block into dbuf

get\_block(fd, ip->i\_block[i], dbuf);

printf(" ino rec\_len name\_len name\n");

dp = (DIR \*)dbuf;

cp = dbuf;

while (cp < dbuf + BLKSIZE) {

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

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

printf("%4d %6d %6d %s\n", dp->inode, dp->rec\_len, dp->name\_len, temp);

cp += dp->rec\_len;

dp = (DIR \*)cp;

}

}

}

char \*disk = "mydisk";

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

{

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDONLY);

if (fd < 0){

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

exit(1);

}

dir();

}

**dir.c output:**

inode\_block = 10

i\_block[0] = 33

ino rec\_len name\_len name

2 12 1 .

2 12 2 ..

11 20 10 lost+found

12 12 4 dir1

13 12 4 dir2

14 12 4 dir3

15 12 4 dir4

16 16 5 file1

17 16 5 file2

18 16 5 file3

19 884 5 file4

//ialloc

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

// define shorter TYPES, save typing efforts

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR; // need this for new version of e2fs

#define BLKSIZE 1024

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

/\*\*\*\*\*\*\*\*\*\* globals \*\*\*\*\*\*\*\*\*\*\*\*\*/

int fd;

int imap, bmap; // IMAP and BMAP block number

int ninodes, nblocks, nfreeInodes, nfreeBlocks;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int put\_block(int fd, int blk, char buf[ ])

{

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

write(fd, buf, BLKSIZE);

}

int tst\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

if (buf[i] & (1 << j))

return 1;

return 0;

}

int set\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

buf[i] |= (1 << j);

}

int clr\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

buf[i] &= ~(1 << j);

}

int decFreeInodes(int dev)

{

char buf[BLKSIZE];

// dec free inodes count in SUPER and GD

get\_block(dev, 1, buf);

sp = (SUPER \*)buf;

sp->s\_free\_inodes\_count--;

put\_block(dev, 1, buf);

get\_block(dev, 2, buf);

gp = (GD \*)buf;

gp->bg\_free\_inodes\_count--;

put\_block(dev, 2, buf);

}

int ialloc(int dev)

{

int i;

char buf[BLKSIZE];

// read inode\_bitmap block

get\_block(dev, imap, buf);

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

if (tst\_bit(buf, i)==0){

set\_bit(buf,i);

decFreeInodes(dev);

put\_block(dev, imap, buf);

return i+1;

}

}

printf("ialloc(): no more free inodes\n");

return 0;

}

char \*disk = "mydisk";

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

{

int i, ino;

char buf[BLKSIZE];

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDWR);

if (fd < 0){

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

exit(1);

}

// read SUPER block

get\_block(fd, 1, buf);

sp = (SUPER \*)buf;

ninodes = sp->s\_inodes\_count;

nblocks = sp->s\_blocks\_count;

nfreeInodes = sp->s\_free\_inodes\_count;

nfreeBlocks = sp->s\_free\_blocks\_count;

printf("ninodes=%d nblocks=%d nfreeInodes=%d nfreeBlocks=%d\n",

ninodes, nblocks, nfreeInodes, nfreeBlocks);

// read Group Descriptor 0

get\_block(fd, 2, buf);

gp = (GD \*)buf;

imap = gp->bg\_inode\_bitmap;

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

getchar();

for (i=0; i < 5; i++){

ino = ialloc(fd);

printf("allocated ino = %d\n", ino);

}

}

**ialloc.c output:**

ninodes=184 nblocks=1440 nfreeInodes=155 nfreeBlocks=1389

imap = 9

//balloc

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ext2fs/ext2\_fs.h>

// define shorter TYPES, save typing efforts

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR; // need this for new version of e2fs

#define BLKSIZE 1024

GD \*gp;

SUPER \*sp;

INODE \*ip;

DIR \*dp;

/\*\*\*\*\*\*\*\*\*\* globals \*\*\*\*\*\*\*\*\*\*\*\*\*/

int fd;

int imap, bmap; // IMAP and BMAP block number

int ninodes, nblocks, nfreeInodes, nfreeBlocks;

int get\_block(int fd, int blk, char buf[ ])

{

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

read(fd, buf, BLKSIZE);

}

int put\_block(int fd, int blk, char buf[ ])

{

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

write(fd, buf, BLKSIZE);

}

int tst\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

if (buf[i] & (1 << j))

return 1;

return 0;

}

int set\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

buf[i] |= (1 << j);

}

int clr\_bit(char \*buf, int bit)

{

int i, j;

i = bit/8; j=bit%8;

buf[i] &= ~(1 << j);

}

int decFreeInodes(int dev)

{

char buf[BLKSIZE];

// dec free inodes count in SUPER and GD

get\_block(dev, 1, buf);

sp = (SUPER \*)buf;

sp->s\_free\_inodes\_count--;

put\_block(dev, 1, buf);

get\_block(dev, 2, buf);

gp = (GD \*)buf;

gp->bg\_free\_inodes\_count--;

put\_block(dev, 2, buf);

}

int balloc(int dev)

{

int i;

char buf[BLKSIZE];

// read inode\_bitmap block

get\_block(dev, bmap, buf);

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

if (tst\_bit(buf, i)==0){

set\_bit(buf,i);

decFreeInodes(dev);

put\_block(dev, bmap, buf);

return i+1;

}

}

printf("balloc(): no more free blocks\n");

return 0;

}

char \*disk = "mydisk";

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

{

int i, bno;

char buf[BLKSIZE];

if (argc > 1)

disk = argv[1];

fd = open(disk, O\_RDWR);

if (fd < 0){

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

exit(1);

}

// read SUPER block

get\_block(fd, 1, buf);

sp = (SUPER \*)buf;

ninodes = sp->s\_inodes\_count;

nblocks = sp->s\_blocks\_count;

nfreeInodes = sp->s\_free\_inodes\_count;

nfreeBlocks = sp->s\_free\_blocks\_count;

printf("ninodes=%d nblocks=%d nfreeInodes=%d nfreeBlocks=%d\n",

ninodes, nblocks, nfreeInodes, nfreeBlocks);

// read Group Descriptor 0

get\_block(fd, 2, buf);

gp = (GD \*)buf;

bmap = gp->bg\_block\_bitmap;

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

getchar();

for (i=0; i < 5; i++){

bno = balloc(fd);

printf("allocated bno = %d\n", bno);

}

}

**balloc.c output:**

ninodes=184 nblocks=1440 nfreeInodes=155 nfreeBlocks=1389

bmap = 8

Header file that contains all necessary functions

#ifndef CONSTANTS

#define CONSTANTS

/\* Debugging Mode \*/

#define DEBUG\_MODE 0

/\* IO \*/

#define BLKSIZE 1024

/\* Define shorter TYPES, save typing efforts \*/

typedef struct ext2\_group\_desc GD;

typedef struct ext2\_super\_block SUPER;

typedef struct ext2\_inode INODE;

typedef struct ext2\_dir\_entry\_2 DIR; // need this for new version of e2fs

/\* inode Constants \*/

#define NUM\_DIRECT\_BLKS 12

/\* Macro -- String Lengths \*/

#define MAX\_FILENAME\_LEN 256

#endif