#include <sys/fcntl.h>

typedef unsigned char u8;

typedef unsigned short u16;

typedef unsigned int u32;

struct partition {

u8 drive; /\* drive number FD=0, HD=0x80, etc. \*/

u8 head; /\* starting head \*/

u8 sector; /\* starting sector \*/

u8 cylinder; /\* starting cylinder \*/

u8 sys\_type; /\* partition type: NTFS, LINUX, etc. \*/

u8 end\_head; /\* end head \*/

u8 end\_sector; /\* end sector \*/

u8 end\_cylinder; /\* end cylinder \*/

u32 start\_sector; /\* starting sector counting from 0 \*/

u32 nr\_sectors; /\* number of of sectors in partition \*/

};

void myfdisk(char\* file);

void myprintf(char \*fmt, ...);

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

{

// PART1: myfdisk

printf("-myfdisk-\n");

myfdisk("vdisk");

// PART2: myprintf

printf("\n-myprintf-\n");

myprintf("Trial 1: %%d=%d %%d=%d %%c=%c %%s=%s %%x=%x %%o=%o\n\n", 123, -321, 'X', "blah", 11111, 1111111111);

myprintf("Trial 2: cha = %c string =%s dec = %d hex = %x oct = %o neg = %d\n\n", 'A', " blahblahblah", 999, 999, 999 -999);

myprintf("Trial 3: argc=%d \n", argc);

for (int i = 0; i < argc; i++)

{myprintf("argv[%d]=%s \n env=%s\n", i, argv[i], \*env);}

return 0;

}

void myfdisk(char \*file)

{

int offset = 0x1BE;

struct partition \*p = 0;

int i = 0;

printf("Device Boot Start End Sectors ID Type\n");

int fd = open(file, O\_RDONLY);

u8 \*buff = malloc(4 \* sizeof(struct partition));

if (fd == -1)

return;

lseek(fd, offset, SEEK\_SET);

read(fd, buff, 4 \* sizeof(struct partition));

p = (struct partition \*)buff;

for (i = 0; i < 4; i++, p++)

printf("%s%d %4d %4d %4d %3x %x\n",

file, i + 1, p->start\_sector, p->start\_sector + p->nr\_sectors-1, p->nr\_sectors, p->sys\_type);

p--;

int p4Start = p->start\_sector;

while (p->start\_sector)

{

int numSectors = p4Start + p->start\_sector;

lseek(fd, numSectors \* 512 + offset, SEEK\_SET);

read(fd, buff, 2 \* sizeof(struct partition));

p = (struct partition \*)buff;

printf("%s%d %4d %4d %4d %3x %x\n", file, i + 1, p->start\_sector + numSectors, p->start\_sector + p->nr\_sectors + numSectors - 1, p->nr\_sectors, p->sys\_type);

p++, i++;

}

}

char \*table = "0123456789ABCDEF";

// prints unsigned u32 recursively, credit: Paul

int rpu(u32 x, int Base)

{

char c;

if (x)

{

c = table[x % Base];

rpu(x / Base, Base);

putchar(c);

}

}

//print unsigned in given base

int printu(u32 x, int Base)

{

if (!x)

putchar('0');

else

rpu(x, Base);

}

//prints an integer (x may be negative!!!)

int printd(int x)

{

if (x < 0){

x = -x;

putchar('-');

}

printu(x, 10);

}

// prints x in HEX (start with 0x )

int printx(u32 x)

{

printu(x, 16);

}

//prints x in Octal (start with 0 )

int printo(u32 x)

{

printu(x, 8);

}

//prints string

int prints(char \*x)

{

while(\*x){

putchar(\*x);

x++;

}

}

//prints string %s, unsigned integer %u, char %c,

//integer %d, unsigned integer in OCT %o, unsigned integer in HEX %x

void myprintf(char \*fmt, ...)

{

u32 \*args = (u32 \*) &fmt + 1;

while (\*fmt)

{

if (\*fmt == '%')

{

fmt++;

switch (\*fmt)

{

case 's':

prints((char\*)\*args);

break;

case 'u':

printu(\*args, 10);

break;

case 'c':

putchar(\*args);

break;

case 'd':

printd(\*args);

break;

case 'o':

printo(\*args);

break;

case 'x':

printx(\*args);

break;

case '%':

putchar('%');

args--;

break;

default:

putchar('%');

putchar(\*fmt);

args--;

break;

}

args++;

}

else

{

putchar(\*fmt);

if(\*fmt == '\n')

putchar('\r');

}

fmt++;

}

}