**CS360 LAB ASSIGNMENT #1**CS360 LAB ASSIGNMENT #1DUE & DEMO: Thursday, 9-2-2021

======================================= Part 1 ========================================

PART 1: myprintf Function

Objective: I/O Function and Stack Usage

READ: Chapter 9: Section 9.8: printf-like function

Given: putchar(char c) of Linux, which prints a char.

**1-1. Write YOUR own prints(char \*s) function to print a string.**

Given: The following printu() function prints an unsigned integer.

typedef unsigned int u32;  
char \*tab = "0123456789ABCDEF";  
int BASE = 10;

int rpu(u32 x)  
{  
 char c;  
 if (x) {  
 c = ctable[x % BASE];  
 rpu(x / BASE);  
 putchar(c);  
 }  
}

int printu(u32 x)  
{  
 (x==0)? putchar('0') : rpu(x);  
 putchar(' ');  
}

EXAMPLE:   
Assume u32 x = 123;  
1st call to rpu(x) : x=123; x%10 = 3 ===> c = tab[3] = '3';   
2nd call to rpu(x) : x=12; x%10 = 2 ===> c = tab[2] = '2';   
3rd call : x=1; x%10 = 1 ===> c = tab[1] = '1';

4th call : x=0 => return ====> print '1';  
 retrun ====> print '2'  
 retrun ====> print '3'

**1-2. Write YOUR OWN fucntions**

int printd(int x) which print an integer (x may be negative!!!)  
int printx(u32 x) which print x in HEX (start with 0x )int printo(u32 x) which print x in Octal (start with 0 )

**3. REQUIREMENTS:**

====================================================================

Write YOUR own myprintf(char \*fmt, ...) function to print:  
char by %c  
string by %s  
unsigned integer by %u  
integer by %d  
unsigned integer in OCT by %o  
unsigned integer in HEX by %x

Ignore field width and precision, just print the items as specified.

**1-3. In the int main(int argc, char \*argv[ ], char \*env[ ]) function, use YOUR myprintf() to print:**

argc value  
argv strings  
env strings

myprintf("cha=%c string=%s dec=%d hex=%x oct=%o neg=%d\n", 'A', "this is a test", 100, 100, 100, -100);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HELP INFO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NOTE: This assignment is for 32-bit GCC, which passes parameters on stack.  
Use “gcc -m32 t.c” to compile your C source files.

int myprintf(char \*fmt, ...) // C compiler requires the 3 DOTs  
{  
 Assume the call is myprintf(fmt, a,b,c,d);  
 Upon entry, the following diagram shows the stack contents.

char \*cp -> "...%c ..%s ..%u .. %d\n"  
HIGH | LOW   
--------------------------- --|--------------------------------------------------  
 | d | c | b | a | fmt |retPC| ebp | locals  
-------------------|---------------------|---------------------------------------  
 | |  
 int \*ip CPU.ebp

1. Let char \*cp point at the format string  
2. Let int \*ip point at the first item to be printed on stack:

NOTE: In 32-bit mode, every entry in the stack is 4-byte;  
 for chars, they are in the lowest byte (of the 4-byte entry)  
 for strings, they are POINTERs to the actual strings.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ALGORITHM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Use cp to scan the format string:  
 spit out each char that's NOT %  
 for each \n, spit out an extra \r

Upon seeing a %: get next char, which must be one of 'c','s','u','d', 'o','x'

Then call YOUR:  
putchar(\*ip) for 'c';  
prints(\*ip) for 's';  
printu(\*ip) for 'u';  
printd(\*ip) for 'd';  
printo(\*ip) for 'o';  
printx(\*ip) for 'x';

Advance ip to point to the next item on stack.

After implementing your myprintf() function, write C program to test your myprintf() function first. Then use it in the assignment.

SAMPLE Solution: ~samples/LAB1/myprintf # download and run it

======================================= Part 2 =======================================

PART 2: Partition Table

**2-1. OBJECTIVES:**Partition table, fdisk, structures in C, open-read files in Unix/Linux

**2-2 Partition Table:**A disk (floppy disk, hard disk, USB drive, SD cards, etc.) consists of 512-bytesectors, which are counted linearly as sector 0,1,2,3,....

A disk is usually divided into several partitions. The partitions are recorded in a partition table at the beginning (the 0th sector) of the disk, called the Master Boot Record (MBR). Inside the MBR, the partition table begins at the byte offset 0x1BE. The Partitin Table contains 4 entries, each 16 bytes long, defined in the following C structure.

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

};

Disk layout  
 S0 S1 S2 ......  
 ------------------------------------------------------  
 |MBR| | | |   
 ------------------------------------------------------

S0 = 512-byte Sector  
 -------------------------------------------------  
 | P1 P2 P3 P4 |  
 --------------------------------|---------------  
 0 0x1BE

heads, sectors, cylinders are for old IDE disks.   
Newer HD, USB, SD use only start\_sector and nr\_sectors.  
So you may ignore the head, sector, cylinder fields of the partition table.

Each partition has a type, which indicates what kind of file system the partition MAY contain. Consult Linux's fdisk to see the partition types.

If a partition is EXTEND type (type=5), the partition's area can be further divided into more partitions. The extended partitions forms a LINK-LIST as the following diagram shows.

------------------------------------------------------------------------------  
Assume P4 is EXT type:  
int extStart = P4's start sector;   
extStart = localMBR  
 E1:P5 start\_sector (r.e. localMBR)  
 E2:next localMBR (r.e. extStart)  
 localMBR  
 E1:P6's start\_sector (r.e. localMBR)  
 E2:next localMBR (r.e. extStart),etc

extStart sector is a localMBR. Each localMBR has a partition table with only 2 entries. The first entry defines the start sector and size of the extended partition. The second entry points to the next localMBR. start\_sector of a partition is RE to its localMBR sector#. All the localMBR sector# are relative to extStart. As usual, the link list ends with a 0.  
-------------------------------------------------------------------------------

Since use fdisk on real hard disks is risky, we shall use a VIRTUAL disk for this assignment. A virtual disk is just a file but its contents are exactly the same as a REAL disk. Download the file ~cs360/samples/LAB1/vdisk to YOUR Linux.

Then run: fdisk vdisk   
 'p' : to see the partition table  
 'q' : to quit fdisk

**REQUIREMENTS**

**2-3. Write a C progrom to display the partition table of vdisk in**

(1). Linux fdisk 'p' output form for the first 4 partitions (%40),  
2). including ALL the extend partitions (%60) <== YOU BETTER DO THIS !!!!

/\* sample code for Part 2 \*/

#include <stdio.h>  
#include <fcntl.h>

#include <sys/types.h>  
#include <unistd.h>

struct partition {  
 // SAME AS GIVEN ABOVE   
};

char \*dev = "vdisk";  
int fd;

// read a disk sector into char buf[512]  
int read\_sector(int fd, int sector, char \*buf)  
{  
 lseek(fd, sector\*512, SEEK\_SET); // lssek to byte sector\*512  
 read(fd, buf, 512); // read 512 bytes into buf[ ]  
}

int main()  
{  
 struct partition \*p;  
 char buf[512];

fd = open(dev, O\_RDONLY); // open dev for READ  
 read\_sector(fd, 0, buf); // read in MBR at sector 0

p = (struct partition \*)(&buf[0x1be]); // p->P1

// print P1's start\_sector, nr\_sectors, sys\_type;

// Write code to print all 4 partitions;

// ASSUME P4 is EXTEND type:  
 Let int extStart = P4's start\_sector;  
 print extStart to see it;

localMBR = extStart;  
loop:  
 read\_sector(fd, localMBR, buf);

// partition table of localMBR in buf[ ] has 2 entries:   
 print entry 1's start\_sector, nr\_sector;  
 compute and print P5's begin, end, nr\_sectors

if (entry 2's start\_sector != 0){  
 compute and print next localMBR sector;  
 continue loop;  
 }  
}

SAMPLE Solution: ~samples/LAB1/ptable  
Download and run it on vdisk to see the outputs