O.S. Lab 3: Simple Disk I/O

Overview to Reading from Disk

We are now ready to begin tackling I/O involving the floppy disk. Specifically we will learn the use of BIOS system calls to read and write an individual disk sector. This will ultimately lead to a routine to read an entire file into memory. Begin by **deleting** the contents of the main() function from the kernel (i.e. the Mad-Libs program), we won't need it again.

Updating Interrupt 33

As before you will now add functionality to the o.s. by writing four new functions. This code should be inserted into **kernel.c** before the *handleInterrupt21()* function. Additionally, once these functions are complete, you will have to add new interrupt calls to access them:

Function	AX=	BX=	CX=	DX=	Action
Read consecutive disk sectors.	**2**	Address of character array where sectors will be stored.	The number of the first sector being read.	Total number of sectors to read.	Call **readSector(bx,cx,dx)**
Write consecutive disk sectors.	**6**	Address of character array storing sectors to write.	The number of the first sector being written.	Total number of sectors to be written.	Call **writeSector(bx,cx,dx)**
Clear screen and set colors	**12**	Background color (1 – 8)	Foreground color (1 – 16)		Call **clearScreen(bx,cx)**

We will now consider the writing of these described functions.

Reading Disk Sectors – Interrupt 19 (0x13)

BIOS interrupt 19 can be used to read or write sectors from the floppy disk. Reading sectors takes the following parameters:

- AH = 2 (this number tells the BIOS to read a sector as opposed to 3 for write)
- **AL = number of sectors to read (up to 255)**
- **BX = address where the data should be stored to (pass your *char** array here)**
- CH = track number (*trackNo*) of starting point

- CL = relative sector number (*relSecNo*) of starting point
- DH = head number (*headNo*) of starting point
- DL = device number (for the floppy disk, use 0)

Recall that AX, CX and DX can be derived by applying the formulas:

- **AX = AH * 256 + AL = 512 + *sectorCount*** in this case;
- **CX = CH * 256 + CL = *trackNo* * 256 + *relSecNo*;** and
- **DX = DH * 256 + DL = *headNo* * 256.**

So this interrupt requires you to know the *relative* sector, head, and track numbers of the first sector you want to read. In this project we are dealing with *absolute* sector numbers. Fortunately, there is a conversion for floppy disks:

- *relSecNo* = (*absSecNo* MOD 18) + 1
- *headNo* = (*absSecNo* DIV 18) MOD 2
- *trackNo* = (*absSecNo* DIV 36)

Recall from the last lab that bcc does not support MOD and DIV so we will have to use the functions we wrote there for these calculations.

You should now have everything you need to write the function **void readSectors(char* buffer, int sector, int sectorCount)** which takes three parameters: a predefined character array of 512 bytes or bigger, an absolute sector number to begin reading at, and the number of sectors to read. Your function should compute the relative sector, head, and track, and call interrupt 19 to read the sector into *buffer*.

To test edit *main()* in **kernel.c** as follows:

```
void main()
{
  char buffer[512];
  makeInterrupt21();
  printLogo();
  interrupt(33,2,buffer,30,1);
  interrupt(33,0,buffer,0,0);
  while (1) ;
}

/* more stuff follows */
```

All we need now is something at sector 30 to read. After compiling **kernel.c** and rebuilding **floppy.img**, execute the command

dd if=msg of=floppya.img bs=512 count=1 seek=30 conv=notrunc

from the Linux command line to place the text file **msg** from the lab web page at disk sector 30. If you run **bochs** and the message prints out, your function works.

Write Disk Sectors – Interrupt 19 (0x13)

Finally create a **writeSectors** function in **kernel.c** to go with the **readSectors** function. Writing sectors is provided by the same BIOS call as reading sectors, and is almost identical. The only difference is that **AX** should equal **768 + *sectorCount*** instead of 512 + … . Your write sectors function should be added to interrupt 33, and should be handled as indicated above. If you implemented **readSectors** correctly, this step will be very simple.

Clear screen/set colors – Interrupt 16 (0x10)/Functions 2 and 6

It's time to clean up a bit of the clutter and have some fun. Write *clearScreen*(*bx*,*cx*) to do the following:

- Issue 24 carriage return/newline combinations. It avoid future complications if you use interrupt 16 calls to place the individual characters;
- Issue the command **interrupt(16,512,0,0,0)**;
- Finally if both *bx* and *cx* are bigger than zero execute

  interrupt(16, 1536, 4096 * (bx – 1) + 256 * (cx – 1), 0, 6223).

Now for some explanation. The first command calls BIOS function 16/2 which repositions the cursor in the upper left-hand corner (coordinates (0,0)) after the existing text has been cleared away. The other interrupt calls BIOS function 16/6 to scroll the window with these parameters:

- AH = 6, indicating function 6;
- AL = 0, meaning scroll whole screen or window;
- **BH = the attribute byte for blank lines**, explained next;
- CH and CL are the row and column for the upper left-hand corner of the window (0,0); and
- DH and DL are the row and column for the lower right-hand corner, (24,79).

The attribute byte describes the color and intensity of subsequent characters (in its four low bits), as well as the background color (in its four high bits). In the *clearScreen()* function call we specify the background color in the parameter *bx*, the character/foreground color in *cx*. Zero values will mean make no change, otherwise specify a color scheme according to the standard 16-color text palette:

Basic colors (Foreground or background)			
0	Black	4	Red
1	Blue	5	Magenta
2	Green	6	Brown
3	Cyan	7	White/light gray

Bright colors (Foreground only)			
8	Gray/dark gray	12	Light red
9	Light blue	13	Light magenta
10	Light green	14	Yellow
11	Light cyan	15	Bright white

If *bx* is larger than 8, or *cx* larger than 16, make no change to the color scheme. (Remember that the values passed in are one larger than those in the table.)

Final Test Case

To test all new functionality edit *main()* in **kernel.c** as follows (changes in **boldface**):

```
void main()
{
  char buffer[512];   int i;
  makeInterrupt21();
  for (i = 0; i < 512; i++) buffer[i] = 0;
  buffer[0] = choice of background color;
  buffer[1] = choice of foreground color;
  interrupt(33,6,buffer,258,1);
  interrupt(33,12,buffer[0]+1,buffer[1]+1,0);
  printLogo();
  interrupt(33,2,buffer,30,1);
  interrupt(33,0,buffer,0,0);
  while (1) ;
}

/* more stuff follows */
```

This creates a configuration file for BlackDOS at sector 258. The first two entries in this file are the foreground and background color, respectively. (These will end up being the only environment variables we use.) Create a unique color scheme for your specific version of BlackDOS. This file creation is followed by clearing the screen and changing the color scheme, before reading and displaying the sample file as before. Finally, since we will need the configuration file for future work, pull it from the disk and save it as a separate file by executing the command

dd if=floppy.img of=config bs=512 skip=258 count=1 conv=notrunc

once from the Linux command line. Do a **hexdump –C config** to make sure you saved this right and don't have a file of all zeros.

Conclusion

When finished, submit a .tar or .zip file (no .rar files) containing all needed files (**bootload**, **bdos.txt**, **kasm.o**, **compileOS.sh**, **kernel.c**, **msg**, **config**, and **README**) to the drop box. For now have **compileOS.sh** add the **msg** file to the disk image; remove this for future labs. **README** should explain what you did and how the t.a. can verify it. Your .zip/.tar file name should be your name.

Last updated 6.27.2019 by T. O'Neil, based on material by M. Black and G. Braught. Previous revisions 1.22.2018, 2.2.2017, 1.25.2016, 1.27.2015, 2.21.2014, 2.22.2014, 11.3.2014.