TABLE OF CONTENTS FRONT PAGE TALKBACK FAQ

"Linux Gazette...*making Linux just a little more fun!*"

Writing Your Own Toy OS (PART II)

By Krishnakumar R.

Part I was published in April.

The next thing that any one should know after learning to make a boot sector and before switching to protected mode is, how to use the BIOS interrupts. BIOS interrupts are the low level routines provided by the BIOS to make the work of the Operating System creator easy. This part of the article would deal with BIOS interrupts.

1. Theory

1.1 Why BIOS ?

BIOS does the copying of the boot sector to the RAM and execution of code there. Besides this there are lot of things that the BIOS does. When an operating system boots up it does not have a video driver or a floppy driver or any other driver as such. To include any such driver in the boot sector is next to impossible. So some other way should be there. The BIOS comes to our help here. BIOS contains various routines we can use. For example there are ready made routines available for various purposes like, checking the equipments installed, controlling the printer, finding out memory size etc. These routines are what we call BIOS interrupts.

1.2 How do we invoke BIOS interrupts ?

In ordinary programming languages we invoke a routine by making a call to the routine. For example in a C program, if there is a routine by name `display` having parameters `noofchar` - number of characters to be displayed, `attr` - attribute of characters displayed is to just to call the routine that is just write the name of the routine. Here we make use of interrupts. That is we make use of assembly instruction `int`.

For example for printing something on the screen we call the C function like this :

```
    display(noofchar, attr);
```

Equivalent to this, when we use BIOS, we write :

```
int 0x10
```

1.3 Now, how do we pass the parameters ?

Before calling the BIOS interrupt, we need to load certain values in prespecified format in the registers. Suppose we are using BIOS interrupt 13h, which is for transferring the data from the floppy to the memory. Before calling interrupt 13h we have to specify the segment address to which the data would be copied. Also we need to pass as parameters the drive number, track number, sector number, number of sectors to be transferred etc. This we do by loading the prespecified registers with the needed values. The idea will be clear after you read the explanation on the boot sector we are going to construct.

One important thing is that the same interrupt can be used for a variety of purposes. The purpose for which a particular interrupt is used depends upon the function number selected. The choice of the function is made depending on the value present in the `ah` register. For example interrupt `13h` can be used for displaying a string as well as for getting the cursor position. If we move value `3` to register `ah` then the function number `3` is selected which is the function used for getting the cursor position. For displaying the string we move `13h` to register `ah` which corresponds to displaying a string on screen.

2. What are we going to do ?

This time our source code consists of two assembly language programs and one C program. First assembly file is the boot sector code. In the boot sector we have written the code to copy the second sector of the floppy to the memory segment `0x500` (the address location is 0x5000). This we do using BIOS interrupt `13h`. The code in the boot sector then transfers control to offset 0 of segment `0x500`. The code in the second assembly file is for displaying a message on screen using BIOS interrupt `10h`. The C program is for transferring the executable code produced from assembly file 1 to boot sector and the executable code produced from the assembly file 2 to the second sector of the floppy.

3. The boot sector

Using interrupt 13h, the boot sector loads the second sector of the floppy into memory location 0x5000 (segment address 0x500). Given below is the source code used for this purpose. Save the code to file `bsect.s`.

```
LOC1=0x500

entry start
start:
        mov ax,#LOC1
        mov es,ax
        mov bx,#0

        mov dl,#0
        mov dh,#0
        mov ch,#0
        mov cl,#2
        mov al,#1
        mov ah,#2

        int 0x13

        jmpi 0,#LOC1
```

The first line is similar to a macro. The next two statements might be familiar to you by now. Then we load the value 0x500 into the `es` register. This is the address location to which the code in the second sector of the floppy (the first sector is the boot sector) is moved to. Now we specify the offset within the segment as zero.

Next we load drive number into `dl` register, head number into `dh` register, track number into `ch` register, sector number into `cl` register and the number of sectors to be transferred to register`al`. So we are going to load the sector 2, of track number 0, drive number 0 to segment `0x500`. All this corresponds to 1.44Mb floppy.

Moving value 2 into register `ah` is corresponds to choosing a function number. This is to choose from the functions provided by the interrupt 13h. We choose function number 2 which is the function used for transferring data from floppy.

Now we call interrupt 13h and finally jump to 0th offset in the segment `0x500`.

4. The second sector

The code in the second sector will as given below :

```
entry start
start:
        mov     ah,#0x03
        xor     bh,bh
        int     0x10

        mov     cx,#26
        mov     bx,#0x0007
        mov     bp,#mymsg
        mov     ax,#0x1301
        int     0x10

loop1:  jmp     loop1

mymsg:
        .byte  13,10
        .ascii "Handling BIOS interrupts"
```

This code will be loaded to segment `0x500` and executed. The code here uses interrupt `10h` to get the current cursor position and then to print a message.

The first three lines of code (starting from the 3rd line) are used to get the current cursor position. Here function number 3 of interrupt `13h` is selected. Then we clear the value in `bh` register. We move the number of characters in the string to register `ch`. To `bx` we move the page number and the attribute that is to be set while displaying. Here we are planning to display white characters on black background. Then address of the message to be be printed in moved to register `bp`. The message consists of two bytes having values 13 and 10 which correspond to an `enter` which is the Carriage Return (CR) and the Line Feed (LF) together. Then there is a 24 character string. Then we select the function which corresponds to printing the string and then moving the cursor. Then comes the call to interrupt. At the end comes the usual loop.

5. The C program

The source code of the C program is given below. Save it into file `write.c`.

```
#include <sys/types.h> /* unistd.h needs this */
#include <unistd.h>    /* contains read/write */
#include <fcntl.h>

int main()
{
                char boot_buf[512];
                int floppy_desc, file_desc;

                file_desc = open("./bsect", O_RDONLY);

                read(file_desc, boot_buf, 510);
                close(file_desc);

                boot_buf[510] = 0x55;
                boot_buf[511] = 0xaa;

                floppy_desc = open("/dev/fd0", O_RDWR);

                lseek(floppy_desc, 0, SEEK_SET);
                write(floppy_desc, boot_buf, 512);

                file_desc = open("./sect2", O_RDONLY);
                read(file_desc, boot_buf, 512);
                close(file_desc);

                lseek(floppy_desc, 512, SEEK_SET);
                write(floppy_desc, boot_buf, 512);

                close(floppy_desc);
}
```

In PART I of this article I had given the description about making the boot floppy. Here there are slight differences. We first copy the file `bsect`, the executable code produced from `bsect.s` to the boot sector. Then we copy the `sect2` the executable corresponding to `sect2.s` the second sector of the floppy. Also the changes to be made to make the floppy bootable have also been performed.

6. Downloads

You can download the sources from

1. bsect.s
2. sect2.s
3. write.c

4. Makefile

Remove the txt extension of the files, and type

```
make
```

at the shell prompt or you can compile everything separately. Type

```
as86 bsect.s -o bsect.o

ld86 -d bsect.o -o bsect
```

and repeat the same for `sect2.s` giving `sect2`. Compile write.c and execute it after putting the boot floppy in to drive by typing :

```
cc write.c -o write
./write
```

7. What Next?

After booting with the floppy you can see the string being displayed. Thus we will have used the BIOS interrupts. In the next part of this series I hope to write about how we can switch the processor to protected mode. Till then, bye !

Krishnakumar R.

Krishnakumar is a final year B.Tech student at Govt. Engg. College Thrissur, Kerala, India. His journey into the land of Operating systems started with module programming in linux . He has built a routing operating system by name GROS.(Details available at his home page: www.askus.way.to) His other interests include Networking, Device drivers, Compiler Porting and Embedded systems.

TABLE OF CONTENTS | FRONT PAGE | TALKBACK | FAQ