HALEOS

A simple x64 bit operating system
Manual/Ref

TABLE OF CONTENTS

Installing/Building

Credit where credits due:
Research references:
 Insight Into the x86-64 Bare PC Application Boot/Load/Run Methodology
StackOverflow (Many users helped answer my questions)
Google Fu

SAXTON HALE OS
Property DAAMAAAAAGGEE!!
-Justin Yang
-Matt Yu

Build Instructions:

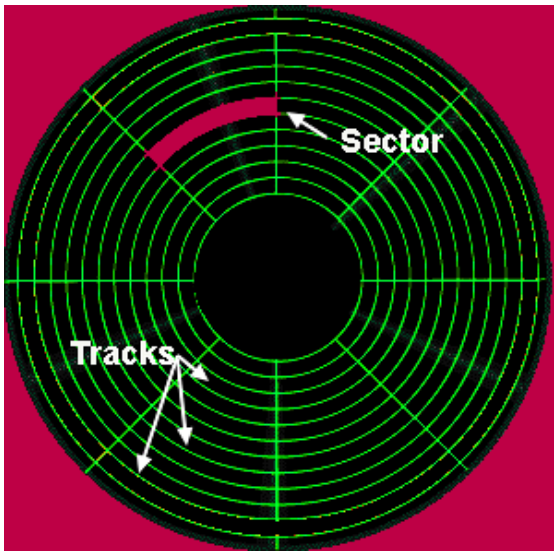
1. Install NASM
2. Run $ make
3. Load “Disk.img” onto VirtualBox, Qemu, or Bosch

Floppy Disk

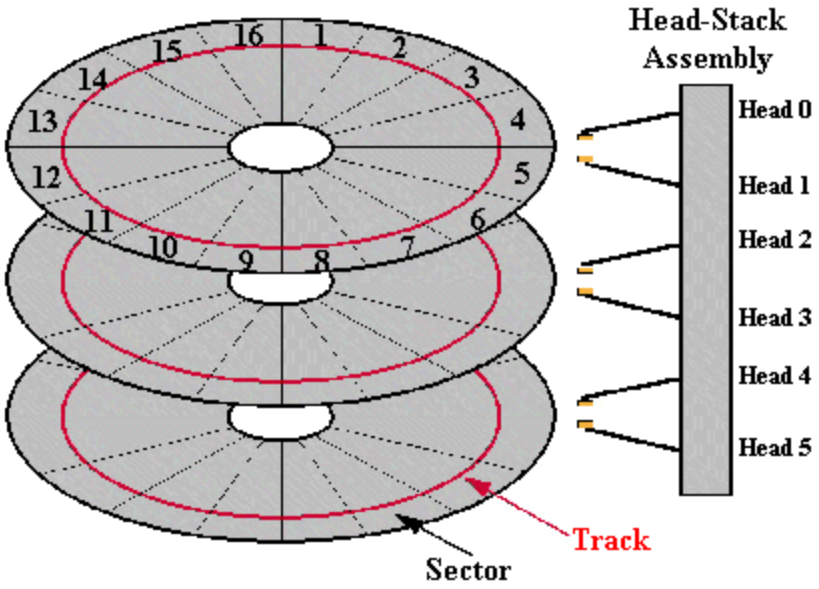
In order to understand how HaleOS is loaded into memory, we must understand how a floppy disk works. A floppy disk has one of the simplest memory layouts, which is what HaleOS will boot from in the beginning development stages. There are four major concepts to a floppy disk. **Sectors, Tracks, Cylinders, and Heads.**

Sectors: A group of 512 bytes. **Sectors are 1 based. Sector 1 is the first sector of the disk.**
Tracks: A collection of sectors.
Head: Side of disk from which the floppy controller reads from. (0,1)
Cylinders: A group of tracks with the same radius on a disk.

These pictures will help you better visualize what sectors, tracks, and cylinders are.



Drive Physical and Logical Organization



We can theoretically calculate the number of sectors that our whole OS image will take up, simply by dividing the size of our binary files by 512. Thus, we can use the Interrupt 0x13 to read that many sectors into a memory location after our boot sector.

INT 13h AH=02h: Read Sectors From Drive

Parameters:

AH	02h
AL	Sectors To Read Count
CH	Cylinder
CL	Sector
DH	Head
DL	Drive
ES:BX	Buffer Address Pointer

Results:

CF	Set On Error, Clear If No Error
AH	Return Code
AL	Actual Sectors Read Count

```
;======================== NEW 998 DISK LOADER ==================================
        mov si, 0x1000
        mov es, si                                                      ; set the start
address of OS image to 0x10000
        mov bx, 0x0000
        mov dh, 5
        mov dl, 0

        ;mov di, word[TOTALREADIMAGE]


        disk_load:
        push dx                         ;Store DX to recall later sectors requested
                                        ;(DH / DL used for INT 0x13 -0x02)
        mov ah, 0x02                    ;Function code for reading
        mov al, dh                      ;Read DH # of sectors
        mov ch, 0x00                    ;Select cylinder 0
        mov dh, 0x00                    ;Select head 0
        mov cl, 0x02                    ;Start from 2nd sector

        int 0x13
        jc DISKREADERROR

        pop dx
        cmp dh, al                      ;Compare the number of times read (AL contains actual
sectors read count)
        jne DISKREADERROR               ;to make sure all Dh sectors have been read
;===============================================================================
```

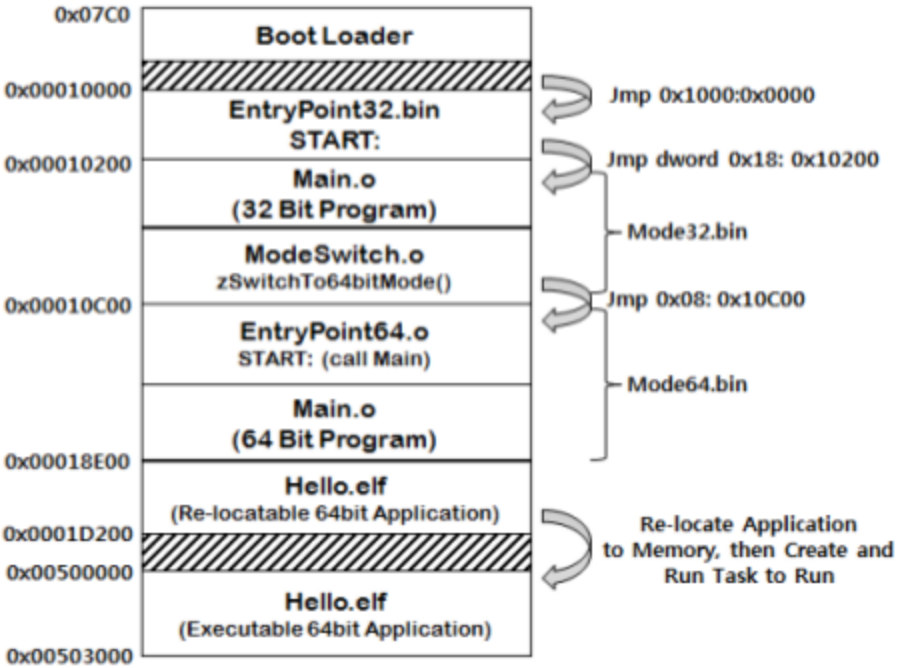
Boot Methodology and Process:

When a PC is booted, it starts in real mode. This limits it to 1MB of memory (address) and allows it access to all BIOS interrupts. The interrupts can be used to accomplish simple tasks that are needed during the boot and load process. This includes setting up and preparing for transition to 32 bit protected mode and finding/loading the kernel from elsewhere on the disk.

Files added by the custom Image Maker are added right after each other. This saves us from having to implement a search function to find our kernel. We just load the sector after our boot sector to the location we specify.

HaleOS loads sectors containing the OS image (11 sectors in total) to/from 0x1000:0x0000 (Physical Address: 0x10000). This is the 1MB address. Sectors loaded from here can now be accessed in protected and 64 bit modes.

At the end of the bootloader sequence, execution jumps to 0x10000:0x0
Where entry.s begins.



Boot Process

[ORG 0x7C00]	This tells the compiler to start code execution at memory location 0x7C00. BIOS loads the bootloader at this location.
[BITS 16]	Tells compiler we are in 16bit mode. All x86 compatible processors boot into 16 bit mode.
[jmp START]	Execution jumps to the START label. (0x7c00:START)
[CLI]	Clear interrupts

Printing to Screen w/ direct memory manipulation

Bootloader manipulates VGA memory to output characters on screen.

(SP: 0xffff)
Push MSG (+2)
Push 0 (+4)
Push 0 (+6)
call PRINTMESSAGE

NOTE: Calling function pushes return address onto stack
Called function pops return address from stack

PRINTMESSAGE: (SP: 0xfff7)
push bp
mov bp, sp

Color has been previously set by CLEARSCREEN.
Note that Bytes 0+1, 2+1, 4+1... have been set with color attribute code.
Bytes 0, 2, 4,.... have been set to zero

Multiply bx by 2 since two bytes make up each character. One for the character and one for the attribute.