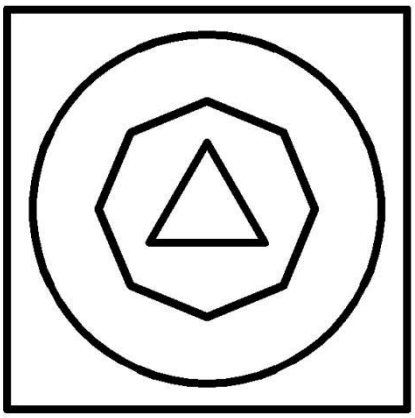
操作系统启动过程



本文件由皇天惊虞制作，免费流通于网络。

制作时间　**2023.7.7**

操作系统启动过程

总结 1

内存固化区
1、以 x86PC 为例，按下开机键后，CPU 处于实模式，内存也啥都没有（除了固化代码，如下图）。首先会寻址内存 0xFFFF0（由硬件设计者事先设置好的），读取内存中的固化代码 ROM，也就是 BIOS 映射区（Basic Input OutPut System）。BIOS 先检查计算机硬件，硬件一切正常后，就会把引导扇区（位于磁盘零磁道零扇区）读入内存 0x7c00，引导扇区的内容就是 bootsect.s（汇编代码，因为汇编代码可以控制每一个内存地址，准确可靠）。

打印开机 logo
2、bootsect.s 首先把自己从内存 0x7c00 的位置挪到 0x90000 的位置，为后续读入 OS 代码腾空间。然后从磁盘将 setup 程序（占 4 个扇区）读入内存 0x90200，然后打印开机 logo（如 system is loading...，logo 可修改，如下图），然后调用 13 号中断，继续读入 system 模块（OS 代码）。bootsect.s 的功能就结束了。bootsect.s 的最后一行跳转到了 setup 所在的 0x90200 位置，开始执行 setup 程序。

部分 setup 程序
3、setup 程序负责完成操作系统启动前的一些设置，它做的事情有：
读了一些硬件参数（如内存大小，光标位置等）
把 system 模块（OS 代码）移到内存的 0 地址处，此后内存 0 地址一直存放 OS 代码。
从实模式变为保护模式，即由 16 位变成 32 位寻址方式（这样能寻址 4G 的内存空间）
最后读取一条高级指令（32 位），跳到 system 模块的位置（0 地址处）开始执行

main.c 进行初始化
4、system 模块的第一个文件是 head.s，它初始化了 gdt(global description table)、页表等数据结构，然后调用了 main.c 文件，进行一堆初始化（如下图），包括内存、中断、设备、时钟、CPU 等内容的初始化。初始化完成后，操作系统就算启动完成了。

总结一下就是，按下开机键后:
先读内存固化区 BIOS，它负责检查硬件然后读入引导扇区
引导扇区负责把 setup 和 system 模块读入内存，并打印开机 logo
setup 读取硬件参数，把 system 模块（OS 代码）移到 0 地址处，并从实模式变为保护模式
正式开始执行 OS 代码，首先执行 head.s 进行一系列初始化操作
初始化完成，启动完成

总结 2

操作系统启动流程

骑行的技术渣
于 2021-11-07 15:48:39 发布
什么是操作系统

操作系统启动过程

操作系统实际是用来操作硬件资源提供给上层应用使用的一种“特殊软件”

操作系统的组成

计算硬件主要组成：CPU ,内存，各种输入输出 IO 设备（显示器，键盘，鼠标，磁盘，网络等外设）

操作系统管理 CPU-----抽象出进程，即 CPU 的管理变成进程的管理

操作系统管理外设-----抽象出文件，即磁盘管理变成文件管理

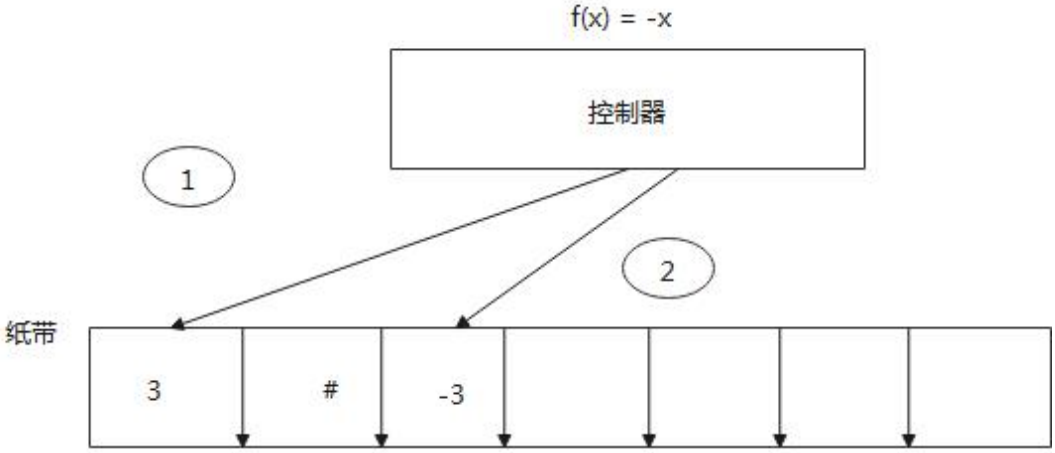
一个操作系统包括四个基本管理模块：进程管理，内存管理，I/O 管理以及文件系统，给上层应用提供的系统接口

计算工作原理

图灵机

计算机最基本的东西是一个计算模型--图灵机

操作系统启动过程



参考人类计算过程，图灵机开始工作后：例如要计算 f(s)

在纸带用编码写下 2

移动“读指针”将 3 读到控制器中

控制器中有一套控制电路对应这个映射，控制电路开始运转，计算出 f(3) = -3,

控制器将结果 -3 写到纸带上。

从图灵机的原理上我们可以看到真实计算机的工作原理就是 取指----执行，再取指---执行，循环操作的一个场景， 这是操作系统中无数次出现的基本场景 取指---执行

操作系统启动过程--第一阶段

bootsect.s 阶段

在按下开机键后，电源接通，计算机加电，在开机启动的时候，计算机首先工作在实模式（16 位寻址模式），硬件电路会初始化设置 PC 寄存器的值。(程序计数器 Program Counter，即程序指令的要执行位置， 是一种通用寄存器，但是有特殊用途，用来指向当前运行指令的下一条指令)，如 IBM 的硬件电路将这个 PC 寄存器的初始值设置为 0xFFFF0 地址， 将这个地址放到地址总线上去，以取出内存中存放的指令，即 0xFFFF0 地址处取出第一条指令开始执行。

那么内存地址 0xFFFF0 上取出的指令是什么？先来了解下 RAM 和 ROM 的概念

内存是随机存储器（random access messory, RAM),属于易失性存储器，故未加电时 RAM 中没有存放任何内容，因此一上电时 RAM 中没有任何信息的。

为了设置一个起点，计算机硬件厂商在只读存储器（read-only memory , ROM)中开辟一块空间，IBM 的 0xFFFF0 就指向这个区域，这段 ROM 就被称为 BIOS（basic input/output system，基本输入输出系统），BIOS 里面放了对基本硬件的测试代码，如对主板，内存等硬件的测试，同时还提供一些让用户调用硬件基本输入输出功能的子程序，如 int 0x10(BIOS 中断)

CPU 从 0xFFFF0 指向 的这段 ROM 中取出的指令要成的工作是测试各种硬件是否正常，如果出现异常则停

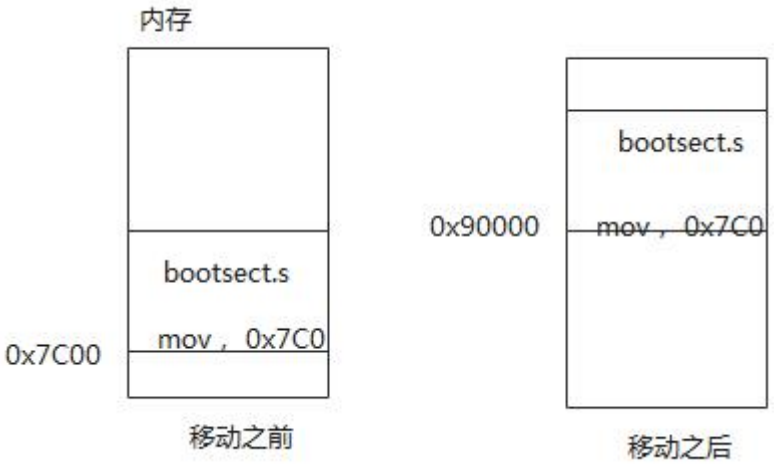
操作系统启动过程

止启动（如 SSD 坏了啦，电脑就会卡那不动或者反复重启），如果检测正常，则利用 BIOS 的输入功能将启动磁盘上启动扇区中的内容读到内存 0x7C00 的地址处（疑问：这里的内存是指 RAM 吗？），并设置 PC 寄存器的地址为 0x7C00

因为 PC 中的地址已经设置为 0x7C00, 接下来就要到该内存地址下执行指令了，该地址下放的是刚才读取的启动扇区的内容（就是启动磁盘上的 0 号柱面，0 号磁头，1 号扇区，共 512 个字节），操作系统第一个要编写的文件就是这个引导扇区中存放的程序代码，通常将其命名为 bootsect.s（是一个汇编文件）

bootsect.s 这个文件中相关代码做的事情如下：

将内存 0x7C00 处的 512 个字节（正好就是 bootsect.s 的全部程序）移动到内存地址 0x90000 开始的一段内存中，这样移动的作用是为读入操作系统核心代码腾出空间



调用 int 0x13 中断（BIOS 中断，这是一个读写磁盘的中断），该中断会调用后的代码用来从磁盘上读入操作系统的 setup.s 文件到内存中， 读取内容是将 0 号驱动器中从 0 号柱面，0 号磁头，2 号扇区开始的 4 个扇区的内容读入到内存中，此处是 0x90200，正好移动到 bootsect.sh 的后面。磁盘上这 4 个扇区中的内容就是操作系统的一个文件，通常称为 setup.s

开机启动过程中打印 LOGO 也是在 bootsect.s 执行的阶段，在该阶段通过 BIOS 中断 in0x10（该中断的作用是在屏幕上输出信息）， 经过一系列寄存器的设置，会取出光标所在的位置，为后面显示纤细做准备
即 LOGO 信息的配置是在这里

接下来就需要从磁盘上读取操作系统的主体部分
在读取主体部分之前，需要先获取一些磁盘参数，比如每个磁盘的扇区个数（通过 BIOS 的 0x13 中断， 该中断调用后能获取每个刺刀的扇区个数）
设置寄存器地址为 0x10000，这个内存地址用来存放从磁盘读出的内容， 该地址是操作系统主体代码在内存中开始的位置（这也是上面 bootsect.s 一开始的代码移动，即从 0x7C00 移动到 0x90000 处的原因，就是为了读入操作系统的主体代码准备的）
上述准备就绪后，开始真正读取系统模块，用一个循环一实现一个磁道一个磁道的读入，同时修改制定寄存器的地址，直达系统模块被全部读入到内存中）

操作系统启动过程

bootsect.s 占用 1 个扇区，setup.s 占用 4 个扇区，故 system 模块从第 6 个扇区开始读

第一阶段启动过程小结

BIOS 读取操作系统的第 1 号扇区 bootsect.s 文件，然后执行 bootsect.s 文件中程序

bootsect.s 继续读取操作系统的 setup.s 文件，将来执行权会交给 setup.s 文件，setup.sh 会完成一些操作系统设置工作

接下里 bootsect.s 还要读入操作系统的主体模板 system,setup.s 执行完成后，会执行 system

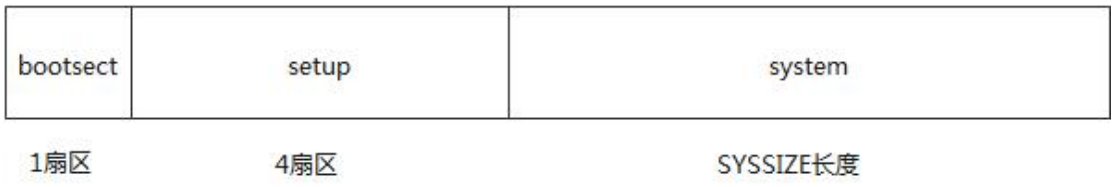
bootsect.s 完成的工作：

将磁盘上从第二到第五这 4 个扇区构成的 setup 的模块读入到内存的 0x90200 处

在显示器上输出操作系统的 logo 标识

从磁盘的第 6 个扇区开始读取长度为 SYSSIZE 长度的操作系统的 system 模块， 并将其放在内存的 0x10000 处

想让 bootsec 工作流程正确，磁盘上的第一个扇区必须防止 bootsect.s 编译后的结果，第二个到第五个的四个扇区必须放置 setup.s 编译后的结果，第六个扇区开始放置编译后的 system 的模块。即启动磁盘上的模块分布必须严格按照如下：



Makefile

为了使操作系统源码文件最后的形式如上图所示，即将操作系统源码编译链接后形成一个二进制文件，也就是著名的操作系统镜像(mirror)文件，需要使用 Makefile，将不同的汇编文件，C 文件， H 文件组织到不同的目录下，控制这些文件的编译来形成一个特定格式的镜像文件

格式如下：

目标：该目标依赖的其他目标
　　产生该目录要执行的命令

操作系统启动过程

操作系统过程---第二阶段

setup 阶段

在引导扇区代码 bootsect.s 的工作：读入 setup ,显示 logo， 读入 system 完成以后，接下来操作系统应该为系统初始化做准备， 即执行 setup（也就是执行 setup.s 中的程序）

因为 bootsect.s 除了要读取 setup 还要读入 system 模块，所有需要在读完 SYSSIZE 长度的 system 模块的内容后 PC 指针跳转到 0x90200 的内存地址，找到 setup.s 的开始位置开始执行（取指---执行的动作）

setup 阶段的工作如下：

初始化必备的一些基本参数需要 setup 阶段获取（如内存有多大，磁盘有多大等）

将来的操作系统要工作的保护模式下（32 位），目前是工作在实模式（16 位），因为 32 位的寻址更多，故要在这里启动保护模式

初始化参数

调用 BIOS 中断 0x15，获取扩展内存的大小， 单位是 KB， 将内存大下的值放在内存地址 0x90000 处，将来系统初始化时可以读取这个值来初始化内存管理

调用 BIOS 中断 0x41 获取硬盘信息，该中断与其他中断有一定区别，如 int 0x10 中断在执行时，会根据中断号到中断向量表（该表存放在内存的 0 地址处）中特定位置取出中断处理地址的入口地址，但中断向量表 0x41 表项存储的却不是中断处理程序的入口地址，而是一个硬盘的基本参数共 16B，表示磁盘包含多少个柱面，多少个磁头，每个磁道多少个扇区的信息

还有其他信息如显示器信息等， 获取方式类似

在获取硬件的基本参数后，setup 的下一项核心工作是启动保护模式

启动保护模式

保护模式，即 32 位模式，与实模式（16 位模）不同的是寻址方式不同（目前还不知道原理），开启 32 位寻址方式后，计算机要启动另一套电路啦解释要执行的指令，即计算 PC 指针的位置？

计算机寻址方式中需要用到两个重要信息，一个是地址表，通常这个表被称为 GDT 表（global descriptor table,GDT 全局描述符表）， 另一个是如何让计算机硬件找到这个表，因为整个寻址过程是自动化的。

setup.s 会完成 GDT 表的构建以及初始化工作，另外，这个表的起始地址会被存放到一个被称为 GDTR 的寄存器中。

GDT 表有三个表项： 表项 0 表示没有用，表项 1 表示操作系统内核代码段，表项 2 表示操作系统内核数据段。

setup.s 在保护模式启动后,取出来的一条指令是 jmpi 0, 8 之前,还会将整个 system 模块拖拽到 0x 地址处(即将内存地址 0x10000~0X90000 间的全部内容移动到地址 0x00000~0x80000 地址处）。

jmpi 跳转的位置即 head.s 的第一句指令，然后就进入操作系统启动的第三阶段

操作系统启动过程-----第三阶段

system 阶段

进入 32 位保护模式以后要执行的一段代码是 head.s。bootsect 将操作系统读入内存，setup 读取一些硬件参数并启动保护模式，我们就可以对操作系统管理的资源的关键数据结构进行初始化。但是初始化之前还有一些准备工作要做：

设置中断表

因为从现在开始操作系统不再使用 BIOS 中断了，实际上将 system 从 0x10000 挪到 0x0 地址处时已经无法使用 BIOS 中断了，因为 BIOS 中断向量表放置在 0 地址处，另外，接管中断是操作系统必须要做的事情，因为不同的操作系统遇到同一中断所实施的操作不同

设置 GDT 表

虽然上面 setup 中设置了 GDT 表，但那时为了 jmpi 0, 8 寻址临时建立的，现在进入 system 模块，需要重新建立

heade.s 中设置 IDT（中断描述符表）表和 GDT 表与在 setup.s 中类似，即设计两个连续 8B 的内存数组，每个数组项占 8B，分别作为中断描述表（interrupt descriptor table ）和 GDT 表

GDT 表的初始位置发生了变化，并且更长，为局部描述符表 local descriptor table LDT 预留空间

设置页表

进入 32 位模式后寻址方式更加复杂，即 GDT[CS] + EIP 算出来的地址仍然不能直接输出到地址总线上，通常还需要用这个地址再去查一次页表才能得到真正的“物理地址” 并输出到地址总线

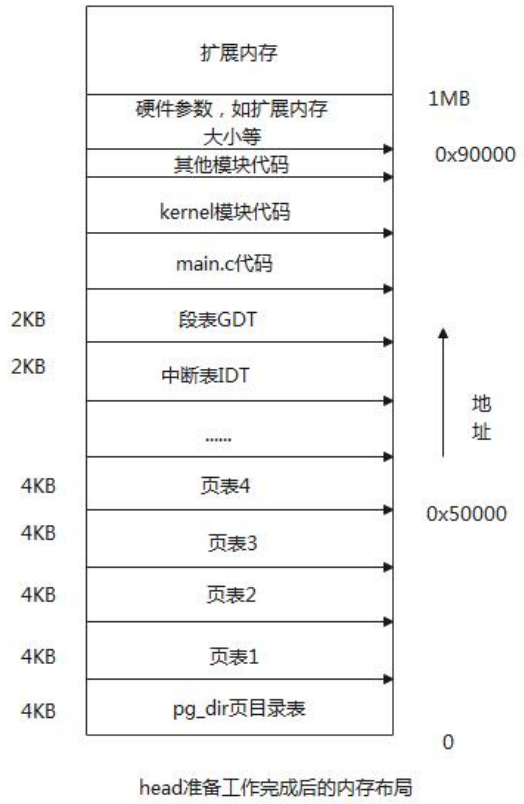
设置页表是在设置 GDT 表完成之后，在页表设置完成后，通过页表映射得到的物理地址就等于输入的地址，即 PageTable

对于操作系统的 system 模块而言，程序中使用的偏移地址和实际输出到物理内存上物理地址实际是一样的，但尽管两个地址一样，每次取出指令，执行指令涉及内存中的操作数时，都要完成整个地址的映射过程（为什么？）

head.s 代码建立这个页表的核心工作主要包括：填写 5 个长度为 4KB 的页表，设置页表寄存器 CR3,以及启动页表电路 三个部分

实际上，操作系统启动涉及的所有工作都是为了形成如下的内存图，执行到这里，这张内存图已经有了，操作系统的准备工作完成，操作系统接下来就可以初始化，head.s 最后一段代码是负责跳转到操作系统的初始化代码处。初始化主要包括：初始化一些数据结构（数组，链表等， 通常用 C 语言）

操作系统启动过程



head准备工作完成后的内存布局

head.s 最后一段代码是完成从汇编语言跳转到 C 程序的代码 L6: jmp L6, 为了保证 main()函数正确返回，需要在跳到 main()之前在栈中压入 main()返回的要执行的指令的地址，而操作系统启动以后不需要返回，因此此处压入标号 L6，在 main()函数返回以后跳翻到 L6 执行，这是一个死循环，所有 main 是一个永远都不能退出的函数，否则计算机就要“死机”

这里就开始进入操作系统启动的第四阶段了

操作系统启动过程----第四阶段

main.c 阶段

一切准备就绪，操作系统终于可以初始化并开始运转，C 语言的 main()函数主要功能就是初始化各种管理软硬件资源的数据结构，通过调用各种初始化函数来完成相应的初始化工作，比如调用 mem_init()来初始化内存，调用 hd_init()来初始化硬盘，其他类似调用对应的*_init()函数。

内存初始化

通过调用 mem_init()函数，该函数有两个参数，分别表示要管理的起始内存地址和结束内存地址，起始地址为 4M，因为 0~-1MB 分配给了系统内核 system， 1M 币~4MB 将分配给磁盘告诉缓存，所以 4MB 以后的才是给用户应用程序可用的。结束地址由 0x90000 处的取出的内容决定（为什么？）

在管理时要需要取整为 4KB 的整数倍，因为操作系统要按照页来管理内存，这里操作系统的一页的大小及时 4KB（不足 1 页的丢弃）

在关键行的各种数据结构初始化完成后，操作系统开始运转

操作系统启动过程

main()函数中最后关键四句代码开始运转操作系统

```
void main(void)

{

.....

sti();

move_to_user_mode();

if(!fork()) {init();}

for(;;){pause();}

}
```

前两条和系统接口相关，第三条和进程有关， 第四条和线程，调度有关，到此，操作系统终于启动，main()函数代码中的 init()会启动一个 shell。

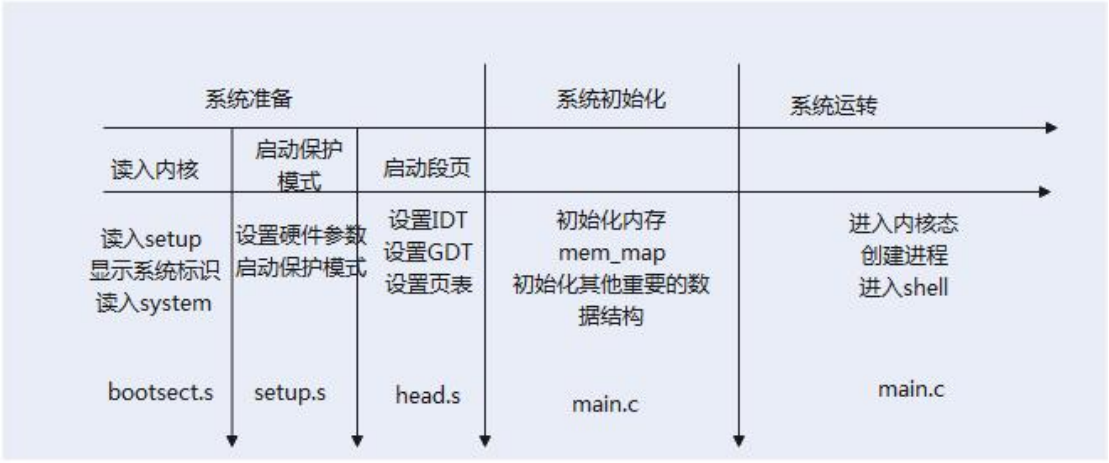
操作系统启动的主要工作就三项：

系统准备：读入内核，启动保护模式，启动段页

系统初始化

系统运转进入 shell

如图



操作系统启动的基本过程

总结 3

操作系统的启动过程详解

黎明 已于 2022-10-12 19:51:09 修改

1.先有鸡还是先有蛋

什么是操作系统？定义有很多，我最喜欢的定义就是：操作系统是管理底层硬件资源，并为上层软件提供服务的软件。说到底操作系统就是一个 C 语言程序，这个 C 语言程序能启动和终止其他软件程序。那么问题来了，操作系统这个 C 程序又是怎么启动的，这个问题很像先有鸡还是先有蛋的问题，好像永远陷入了死循环。

2.什么是 MBR

想要知道答案还得要了解一些计算机组成原理的知识。第一个要解决的问题是什么是 MBR。

先来看看磁盘的组成原理。磁盘的组成主要有盘片、机械手臂、磁头与主轴马达所组成， 而数据的写入其实是在盘片上面。盘片上面又可细分出扇区与磁道两种单位， 其中扇区的物理量设计有两种大小，分别是 512 字节与 4K 字节（高级磁盘）。那么是否每个扇区都一样重要呢？其实整个磁盘的第一个扇区特别的重要，因为他记录了整个磁盘的重要信息，第一个扇区也叫主引导扇区。它在硬盘上的三维地址为（磁盘，磁道，扇区）这个三元组。这个三元组和我们的逻辑地址会有一个一对一的映射关系。

主引导扇区由三部分构成：

主引导记录（Master Boot Record, MBR）：可以安装开机管理程序，有 446 字节。
磁盘分区表（Disk Partition Table, DPT）：记录整个磁盘分区状态，有 64 字节。
主引导扇区结束标志（MN） 0XAA55, 有 2 字节 。
446+64+2=512 字节。所以 MBR（Master Boot Record）主引导记录的作用就是安装启动引导程序的地方，并且 MBR 的位置就是（0，0，1）。

这里是一些扩展部分，可以不看。
由于分区表所在区块仅有 64 字节容量，因此最多仅能有四组记录区，每组记录区记录了该区段的启始与结束的柱面号码，四个分区的大小可以不一样， 这四个分区的被称为主要分区或扩展分区。重点来了，第一个分区存放了主引导扇区的内容和一些数据内容，这没话说。那么就还剩三个分区可以使用，如果第二个分区和第三个分区被使用后，那么最后一个分区就是扩展分区。注意主要分区可以被格式化，但是扩展分区不能被格式化。扩展分区可以分成很多更小的分区，这些小分区叫逻辑分区，逻辑分区可以被格式化。

3.BIOS 基本原理

BIOS 是一个写入到主板上的一个固件（ 固件就是写入到硬件上的一个软件程序）。BIOS 就是在开机的时候，计算机系统会主动执行的第一个程序。接下来 BIOS 会去分析计算机里面有哪些硬件设备，并进行硬件自检，如果硬件接口不匹配就会报错。之后 BIOS 会在硬盘中查找启动设备，并且到硬盘里面去读取第一个扇区的 MBR 位置。 MBR 这个仅有 446 字节的硬盘容量里面会放置最基本的开机管理程序，此时 BIOS 就功成圆满，而接下来就是 MBR 内的开机管理程序的工作了。这个开机管理程序的目的是在加载核心文件，由于开机管理程序是操作系统在安装的时候所提供的，所以他会认识硬盘内的文件系统格式，因此就能够读取核心文

件。这个核心文件就是操作系统内核， 然后接下来核心文件加载成功意味着操作系统成功启动，BIOS 将硬件的控制权交给操作系统。

4.BIOS 深入理解

看完前面的 BIOS 基本原理，大部分人基本上还是一头雾水，为什么 BIOS 就是计算机执行的第一个程序？BIOS 这个程序里面写了什么？等等一系列问题。

要解释这些问题需要有一些前置知识，我总不可能从原子组成分子开始讲起吧。这些前置知识就是：

内存是存储数据的地方，给出一个地址信号，内存可以返回该地址所对应的数据。
CPU 的工作方式就是不断从内存中取出指令，并执行，是一个无情的执行指令的机器。
CPU 从内存的哪个地址取出指令，是由一个寄存器中的值决定的，这个值会不断进行 “+1” 操作，或者由某条跳转指令指定其值是多少。
以上知识就是计算机组成原理的一些基本知识，只需要知道这三点前置知识，你就能专业地解释计算机的启动过程了。

先来理解一下内存映射，CPU 地址总线的宽度决定了可访问的内存空间的大小。32 位的 CPU 地址总线宽度为 32 位，地址范围是 4G。64 位的 CPU 地址总线宽度为 64 位，理论上地址范围是 16384PB 或 16777216TB。但实际上很多 64 位 CPU 使用 40 位地址线,最大寻址空间仅为 1TB。

但是可访问的内存空间这么大，并不等于说全都给内存使用，也就是说寻址的对象不只有内存，还有一些外设也要通过地址总线的方式去访问，那怎么去访问这些外设呢？就是在地址范围中划出一片片的区域，这块给显存使用，那块给硬盘控制器使用，等等 。就相当于在显存等外设的相应位置上读取或者写入，就好像这些外设的存储区域，被映射到了内存中的某一片区域一样。这样我们就不用管那些外设，关注点仍然是一个简简单单的内存。这就是所谓的内存映射。

小总结：就是当你按下电源键的那一刻，整台计算机的硬件开始通电，硬件上的固件就映射到内存当中。以磁盘为例，磁盘上的 BIOS 这个固件被映射到的内存位置为 0xC0000 - 0xFFFFF 。0xFFFFF-0xC0000=256kB。这个 256KB 的固件程序包括了显卡 BIOS、IDE 控制器 BIOS 和最重要的系统 BIOS。其中，系统 BIOS 在内存的 0xF0000 - 0xFFFFF 位置。

我们要用到另一个前置知识了，就是 CPU 从内存的哪个位置取出执行并执行呢？是 PC 寄存器中的地址值。BIOS 程序的入口地址也就是开始地址是 0xFFFF0（人家就那么写的），也就是开机键一按下，将 pc 寄存器中的值变成 0xFFFF0，然后 CPU 就开始马不停蹄地跑了起来。

小总结：在你开机的一瞬间，CPU 通电之后，CPU 的 PC 寄存器被强制初始化为 0xFFFF0。如果再说具体些，CPU 将段基址寄存器 cs 初始化为 0xF000，将偏移地址寄存器 IP 初始化为 0xFFF0，根据实模式下的最终地址计算规则，将段基址左移 4 位，加上偏移地址，得到最终的物理地址也就是抽象出来的 PC 寄存器地址为 0xFFFF0。

上面就分析完了 BIOS 程序是如何映射到内存以及 CPU 是如何知道 BIOS 程序在内存中的位置。

那接下来的问题似乎也非常自然地就问出来了，那就是 BIOS 程序里到底写了啥？我们先来简单分析一下，

你看系统 BIOS 程序的入口地址是 0xFFFF0，说明程序是从这执行的。实模式下内存的下边界就是 0xFFFFF，也就是只剩下 16 个字节的空间可以写代码了，这够干啥的呢？如果你有心的话应该能猜出，入口地址处可能是个跳转指令，跳到一个更大范围的空间去执行自己的任务。没错就是这样，0xFFFF0 处存储的机器指令，翻译成汇编语言是：

```
jmp far f000:e05b
1
```

意思是跳转到物理地址 0xfe05b 处开始执行。地址 0xfe05b 处开始，便是 BIOS 真正发挥作用的代码了，这块代码会检测一些外设信息，并初始化好硬件，建立中断向量表并填写中断例程。这里的部分不要展开，这只是一段写死的程序而已，而且对理解开机启动过程无帮助，我们看后面精彩的部分，也就是 BIOS 的最后一项工作：加载启动区。

5. 0x7c00 是啥

加载在计算机领域就是指，把某设备上（比如硬盘）的程序复制到内存中的过程。那加载启动区这个过程，翻译过来就是，BIOS 程序把启动区的内容复制到了内存中的某个区域。好了，问题又自然出来了，启动区是哪里？被复制到了内存的哪个位置？

BIOS 会按照顺序，读取这些启动盘中位于 0 盘 0 道 1 扇区的内容。 这 0 盘 0 道 1 扇区的内容一共有 512 个字节，如果末尾的两个字节分别是 0x55 和 0xAA，那么 BIOS 就会认为它是个启动区。如果不是，那么按顺序继续向下个设备中寻找位于 0 盘 0 道 1 扇区的内容。如果最后发现都没找到符合条件的，那直接报出一个无启动区的错误。

注释：看看第二段的内容，有一句话是“主引导扇区结束标志（MN） 0XAA55, 有 2 字节”，但刚刚又说“BIOS 如果读取到末尾的两个字节分别是 0x55 和 0xAA,那么 BIOS 就会认为它是个启动区。”好像写反了，但其实并没有，因为机器使用的是小端法。现在 Intel 和 arm 系列的处理器使用的是小端法；IBM,sun 系列的处理器使用的是大端法。

BIOS 找到了这个启动区之后干嘛呢？当然是把这 512 个字节的内容，一个比特都不少的全部复制到内存的 0x7c00 这个位置。启动区内容此时已经被 BIOS 程序复制到了内存的 0x7c00 这个位置，然后呢？这个其实也不难猜测，启动区的内容就是我们自己写的代码了，复制到这里之后，就开始执行呗，之后我们的程序就接管了接下来的流程，BIOS 的使命也就结束了。所以复制完之后，接下来应该是一个跳转指令，PC 寄存器的值变为 0x7c00，指令开始从这里执行。

对了，现在似乎就剩下一个问题了，为什么非要是 0x7c00 呢？这个问题答案也很简单，那就是人家 BIOS 开发团队就是这样定的，之后也不好改了，不然不兼容。正因为 BIOS 将启动区的代码加载到了 0x7c00，因此有了一个偏移量，所以所有写启动区代码的人就需要在开头写死一个这样的代码，不然全都串位了。然后正因为所有写操作系统的，启动区的第一行汇编代码都写死了这个数字，那 BIOS 开发者最初定的这个数字就不好改了，否则它得挨个联系各个操作系统的开发厂商，说唉我这个地址改一下哈，你们跟着改改。在公司推动另一个团队改个代码都得大费周折，想想看这样的推动得耗费多大人力。况且即使改了，之前的代码也都不兼容了。

所以 BIOS 负责加载了启动区，而启动区又负责加载真正的操作系统内核。

6. 总结

操作系统启动过程

现在经过好几轮跳跳跳，终于跳到内核代码，我们来回顾一下：

按下开机键，CPU 将 PC 寄存器的值强制初始化为 0xffff0，这个位置是 BIOS 程序的入口地址（一跳）
该入口地址处是一个跳转指令，跳转到 0xfe05b 位置，开始执行（二跳）
执行了一些硬件检测工作后，最后一步将启动区内容加载到内存 0x7c00，并跳转到这里（三跳）
启动区代码主要是加载操作系统内核，并跳转到加载处（四跳）
经过这连续的四次跳跃，操作系统就成功启动了。终于来到了操作系统的世界了，剩下的内容，可以说是整个操作系统课程所讲述的原理，分段、分页、建立中断、设备驱动、内存管理、进程管理、文件系统、用户态接口、TCP\IP 协议栈等等。

总结 4

操作系统的启动过程 | Windows、嵌入式系统、Linux

☞ 通用计算机启动过程

1 一个基础固件：BIOS

一个基础固件：BIOS→基本 IO 系统，它提供以下功能：

- 上电后自检功能 Power-On Self-Test，即 POST：上电后，识别硬件配置并对其进行自检，保证正常运行和初始化；
- 基本 IO 驱动与事件处理功能：初始化并驱动硬件，如显示器、串口、键盘等接口，使能基本的中断；
- 启动参数设置功能：过程中允许通过热键启动设置界面，进而对 CMOS RAM 中的启动参数进行配置。CMOS RAM 等效于 BBSRAM，存放启动配置数据，电池掉电后数据丢失；
- 系统自动装载功能：在系统自检成功后，根据启动顺序，将相应启动设备主引导记录 MBR（一般位于 0 磁道的 0 扇区，大小为 512 字节）的引导程序装入内存并从入口地址运行；

2 WIndows 操作系统启动过程

① 系统上电或复位，X86 处理器复位——代码段寄存器 CS 为 0xFFFF，指令指针寄存器 IP 为）0x0000，CS:IP 地址存放下一条跳转指令，跳转至 ROM 中 BIOS 入口地址 0xFFFFFFF0（复位向量地址），并启动 BIOS。

② BIOS 上电自检，若出现错误则初始化基本硬件，允许用户进行参数配置。

③ BIOS 将第 1 个启动设备的第 1 个扇区加载到系统 RAM 的 0x7C000 地址，启动 MBR 中的引导程序，进入引导的第一个阶段。

④ 调用 Windows MBR Loader 或 LILO GRUB WinGrub 等引导程序，进入引导的第二个阶段。

⑤ 引导程序调用 OS Boot Loader 把用户选择的操作系统内核加载到内存，并跳转到操作系统入口地址开始执行。

此时，计算机的控制权交给了操作系统，基本启动过程完成。

☞ 嵌入式系统启动过程

MCS-51 MCU 上电复位后，PC 寄存器的初值为 0x0000。0000H，0001H，0002H 这三个单元存放了一条无条件跳转指令，当从该地址执行时将直接跳转到主程序的入口地址。

arm 处理器复位后将从 0x00000000 地址处开始执行指令。

1 处理器片内集成启动固件——嵌入式系统启动

① 上电程序引导

片内集成独立 Boot ROM (Brom)，代码 Rom Boot Loader (RBL) 支持从 NAND Flash、SPI、UART 等外部接口启动。

- 加电后，处理器将从该 ROM 的复位向量地址开始执行，RBL 通过判断处理器特定引脚的电平来进入正常启动模式或开发模式；
- RBL 获取下一步要执行的代码，并将其复制到 SRAM 或 SDRAM 中引导执行；

② 嵌入式操作系统装载

嵌入式操作系统的引导也需要特定的机制和软件支持，即 Boot Loader。

- 复位后，将 Boot Loader 代码从 Flash 拷贝到 SDRAM 的特定区间并引导执行；
- Boot Loader 程序执行一系列基本的硬件初始化工作；
- 将自身拷贝到 SDRAM 中，RAM 中的 Boot Loader 继续执行，为操作系统的运行做好环境准备，并将外部存储器中的操作系统内核映像及根文件系统映像拷贝到内存中的代码，数据空间，设置内核启动参数；
- 跳转至内核入口地址开始执行。

2 裸机

直接部署在嵌入式硬件上的软件称为裸机代码，或裸机（应用）软件。一般来说都是些无限循环结构，比如空调、冰箱等嵌入式系统。通过板级支持包 BSP，向下屏蔽硬件的细节，向上提供统一的服务和接口。没有操作系统，通过中断来实现多任务运行。

☞ Linux 系统启动过程

计算机接通电源上电后，需要经过 BIOS 加电自检、MBR 系统引导、加载内核三步之后，操作系统才会启动。

1 BIOS 加电自检

- BIOS 全称 Basic Input/Output System，即基本输入输出系统，它是一个被永久刻录在 ROM 中的软件，加电自检是指 Power On Self Test，POST，属于 BIOS 的主要组成部分。
- 计算机在接通电源后，BIOS 通过 POST 来加载硬件信息，进行内存、CPU、主板等检测，如果硬件设备正常工作，BIOS 会寻找硬盘第一个扇区中存储的数据，并使用 MBR 中的数据激活引导加载程序。

2 MBR 系统引导

- MBR 全程 Master Boot Recode，是一种磁盘分区格式，也是以此种格式的磁盘中 0 盘片 0 扇区中存储的一段记录——主引导记录。磁盘中扇区的大小为 512byte，主引导记录 MBR 占据第一个扇区的前 446 字节，剩余的空间依次存储一个 64 字节的磁盘分区表，和一个用于标识 MBR 是否有效的 2 字节的模数。
- 主引导记录 MBR 中包含一个实现引导加载功能的程序——Boot Loader。由于 BIOS 只能访问很少量的数据，所以 MBR 中的引导加载程序其实只是一段初始程序的加载程序 Initial Program Loader，IPL，这段程序唯一的功能就是定位并加载 Boot Loader 的主体程序。
- 加载引导分为两个阶段
- - 第一阶段，BIOS 引导 IPL 获取 Boot Loader 主题程序在磁盘中的位置，此时系统启动的控制权由 BIOS 转移到 MBR；
- - 第二阶段，Boot Loader 主题程序与操作系统对应的内核，定位到内核文件所在的位置，并将其加载到计算机内存中，此时系统启动的控制权由 MBR 转移到内核。

3 加载内核

- 内核是操作系统的核心，Linux 操作系统的内核就是 Linux。内核以一种自解压的压缩格式压缩，它与一个初始化的内存映像和存储设备映像表一起存储在 /boot 目录下。
- 在选定的内核被加载到内存中并开始执行前需要先从压缩格式中解压，一旦内核自解压完成，systemd 进程（也就是早期版本中的 init 进程）便被启动。

- systemd 进程的启动标识着引导过程的结束，也标识着启动过程的开始。在系统启动之初，由于系统中没有除 systemd 之外的程序执行，系统初始化工作尚未完成，因此计算机不能执行任何和用户相关的功能性工作。
- 系统初始化需要进行挂载文件系统、启动后台服务等等一系列工作，这些初始化工作全部由 systemd 进程完成。对于用户来说，系统初始化完成后，系统才算正式启动。

4 附：init 进程启动级别

系统启动流程：

BIOS → MBR → boot loader → kernel → init

BIOS → MBR → GRUB → kernel → init

- BIOS：找到启动介质 - 移动硬盘、磁盘、U 盘等，找到启动介质后读取其中的第一个扇区；
- MBR：第一个扇区（512 字节）称为主引导记录。主引导记录分为 3 部分，前 446byte 是引导信息，后 64byte 是磁盘分区信息，最后 2byte 是标志位。MBR 的作用是找到 boot loader 。
- GRUP：是一种 boot loader ，用于加载 kernel 核心信息。
- kernel：内核。
- init：内核的第一个程序，分为 7 个启动级别。

查看启动级别配置文件

inti 命令可以切换系统的启动级别

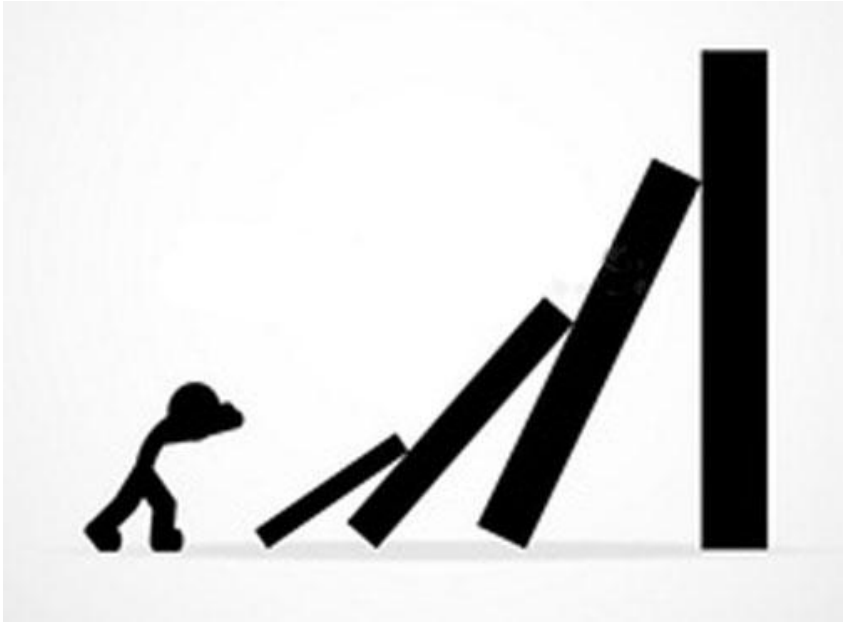
```
inti 0/1/2/3/4/5/6
```

- 0 表示关机（不能设置为开机默认启动级别）
- 1 表示单用户
- 2 表示多用户（无网络的 3 级别）
- 3 多用户（命令行模式，字符终端）
- 4 用于开发
- 5 图形界面，默认启动方式
- 6reboot（不能设置为开机默认启动级别）

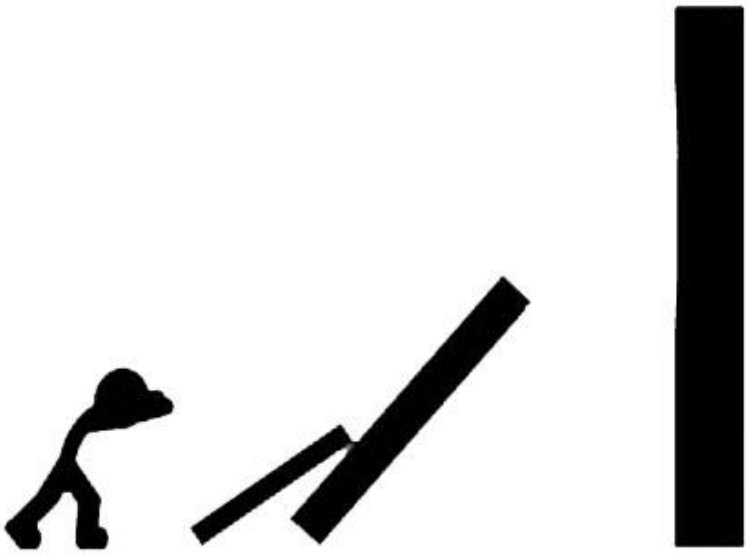
```
runlevel   #查看系统的启动级别
```

总结 5

如下图积木一级一级的形成骨牌效应，从最小的积木到最后的一个最大积木，一个一个的向前推最终全都倒下。



如下图第三个积木的缺失，导致最后的积木无法倒下。



操作系统启动过程，就如同上面的积木一样，（20 位地址）按下开机键，CPU 将 PC 寄存器的值强制初始化为 0xffff0，这个位置是 BIOS 程序的入口地址（一跳）对应第一个积木，该入口地址处是一个跳转指令，跳转到 0xfe05b 位置，开始执行（二跳）对应第 2 个积木，执行了一些硬件检测工作后，最后一步将启动区内容加载到内存 0x7c00，并跳转到这里（三跳）对应第 3 个积木，启动区代码主要是加载操作系统内核，并跳转到加载处（四跳）对应第 4 个积木。