**一、BIOS**

计算机加电后，特殊的硬件电路在CPU的一个引脚上产生RESET逻辑值，把处理器的一些寄存器设置成固定的值（包括CS和EIP），并执行在物理地址0xfffffff0处找到的代码。硬件把这个地址映射到ROM种。（ROM中所存放的程序集传统上称为BIOS）

系统一旦被初始化就不再使用BIOS，而是为计算机上的每个硬件设备提供各自的设备驱动程序。

BIOS使用实模式的地址。一个实模式的地址有一个seg段和一个off偏移量组成，相应的物理地址计算式为：seg\*16+off。同时对GDT，LDT和页表的初始化代码必须在实模式下运行。

BIOS启动过程实际上执行以下4个操作：

1.对计算机硬件执行一系列测试，用来检测现在都有什么设备以及这些设备是否正常工作。这个步骤通常称为POST（上电自检）。

2.初始化硬件设备，保证所有硬件设备操作不会引起IRO线与I/O端口冲突。在步骤最后会显示系统中所有安装的PCI设备的一个列表。

3.搜索一个操作系统来启动。实际上根据BIOS的设置，这个过程可能要试图访问（按照用户预定义的次序）系统中的软盘，硬盘，和CD-ROM的第一个扇区（引导扇区）。

4.只要找到一个有效的设备，就把第一个扇区的内容拷贝到RAM中从物理地址0x00007c00开始的位置，然后跳转到这个地址处，开始执行装载进来的代码。

**主引导区从0x00007c00的原因：**

0x7C00这个地址来自Intel的第一代个人电脑芯片8088，以后的CPU为了保持兼容，一直使用这个地址。 1981年8月，IBM公司最早的个人电脑IBM PC 5150上市，就用了这个芯片。当时，搭配的操作系统是86-DOS。这个操作系统需要的内存最少是32KB。我们知道，内存地址从0x0000开始编号，32KB的内存就是0x0000～0x7FFF。 8088芯片本身需要占用0x0000～0x03FF，用来保存各种中断处理程序的储存位置。（主引导记录本身就是中断信号INT 19h的处理程序。）所以，内存只剩下0x0400～0x7FFF可以使用。 为了把尽量多的连续内存留给操作系统，主引导记录就被放到了内存地址的尾部。由于一个扇区是512字节，主引导记录本身也会产生数据，需要另外留出512字节保存。所以，它的预留位置就变成了：

0x7FFF - 512 - 512 + 1 = 0x7C00

+--------------------- 0x0

| Interrupts vectors

+--------------------- 0x400

| BIOS data area

+--------------------- 0x5??

| OS load area

+--------------------- 0x7C00

| Boot sector

+--------------------- 0x7E00

| Boot data/stack

+--------------------- 0x7FFF

| (not used)

+--------------------- (...)

**二、引导装入程序**

引导装入程序（boot loader）是由BIOS用来把操作系统的内核映像装载到RAM中所调用的一个程序。

**从软盘启动**

从软盘启动，必须把第一个扇区中所存放的指令装载到RAM中并执行；这些指令再把包含内核映像的其他所有扇区都拷贝到RAM中。

当BIOS装载软件的第一个扇区时，就是拷贝引导装入程序的代码。BIOS通过跳转到物理地址的0x00007c00来调用引导装入程序，装入程序执行以下操作。

1.把装入程序从0X00007C00处拷贝到0X00090000处

2.设置从0x00003ff4开始的实模式堆栈。这个堆栈会向低地址方向延伸。

3.建立磁盘参数表，这是BIOS用来处理软盘设备驱动程序的。

4.调用一个BIOS过程显示Loading信息

5.调用一个BIOS过程从软盘中装载内核映像的setup（）代码，并把该代码放入从0x00090200开始的RAM中。

6.调用一个BIOS过程从软盘中装载其余内核的映像，并把内核映像放入从低地址到0x00010000（适用于使用make zImage编译的小内核映像）或者高地址0x00100000（适用使用make bzImage编译的大内核映像）开始的RAM中。

7.跳转到SetUp（）代码。

**从硬盘启动**

大多数情况下，Linux是从硬盘装入的，并需要一个两阶段的引导装入程序。Linux最常使用的引导装入程序就是LILO（Linux Loader）。

MBR或者分区引导扇区包括一个小的引导装入程序，由BIOS把这个小程序装入地址0x00007c00开始的RAM中。这个小程序把自己移动到地址0x0009a000，建立实模式栈（从0x0009b000到0x0009a200），并把LILO的第二部分装入到地址0x0009b000开始的RAM中。

第二部分又依次从磁盘读取可用的操作系统映射表，并提供给用户一个提示符。用户选择被装入的内核后，引导装入程序把相应分区的引导扇区拷贝到RAM中并执行它，或者直接把内核映像拷贝到RAM处。

**三、SetUp（）函数**

连接程序把setup（）汇编语言函数代码放在紧跟内核集成的引导装入程序之后，即内核映像文件的 偏移量0x200处。同时把它拷贝到物理地址0x00090200开始的RAM中。

Setup（）函数必须初始化计算机中的硬件设备并为内核程序的执行建立环境。虽然BIOS已经初始化大部分硬件设备。但Linux并不依赖于BIOS，而是以自己的方式重新初始化设备以增加可移植性和健壮性。

Setup（）本质上执行以下操作：

1.调用一个BIOS过程来检查系统中可用RAM的数量。

2.设置键盘重复延时和速率

3.初始化视频卡

4.重新初始化磁盘控制器并检测硬盘参数。

5.检查IBM微通道总线

6.检查PS/2设备

7.检查对高级电源管理BIOS的支持

8.如果内核映像被低装载到RAM中（在物理地址0x00010000处），就把它移动到物理地址0x00001000处，反之，如果内核映像被高装载到RAM中，就不用移动。

（这个步骤为了能在软盘上存储内核映像并节省启动时间，存放在磁盘上的内核映像是压缩的，解压程序需要一些空闲空间作为临时缓冲区（紧挨着RAM中的内核映像））

9.建立一个临时中断描述符表（IDT）和一个临时全局描述符表（GDT）

10.如果需要，重置浮点单元。

11.重新编写可编程中断控制器（PIC），并把16个硬件中断映射到32到47的中断向量。

（内核必须执行该步骤，因为BIOS错误地把硬件中断映射到从0到15的中断向量，而这些中断向量早已用于CPU的异常）

12.通过设置cr0状态寄存器中的PE位把cpu从实模式切换到保护模式。

13.跳转到startup\_32（）汇编语言函数。

**四、startup\_32（）函数**

有两个不同的startup\_32函数。

第一个函数执行：

1.初始化段寄存器和一个临时堆栈。

2.用0填充由\_edata和\_end符号标识的内核未初始化数据区。

3.调用decompress\_kernel（）函数来解压内核映像。如果内核映像是低装载的，解压后的内核就被放在物理地址0x00100000处。否则，如果内核映像是高装载的，那么解压后的内核就被放在位于这个内核映像之后的临时缓冲区。然后解压后的映像就被移动到最终的从物理地址0x00100000开始的位置。

4.跳转到物理地址0x0010000处。

第二个函数执行：（实际上为第一个Linux进程建立执行环境）

1.把段寄存器初始化为最终值。

2.为进程0建立内核态堆栈。

3.调用setup\_idt()，用空中断处理程序填充IDT

4.把从BIOS中获得的系统参数和传递给操作系统的参数放入第一个页框中。

5.识别处理器模式

6.用GDT和IDT表地址来填充gdtr和idtr寄存器

7.跳转到start\_kernel()函数。

**五、start\_kernel()函数**

start\_kernel()函数完成Linux内核的初始化工作。