linux0.01(原始版)源代码分析笔记

1. 整体结构：

第一个文件夹boot ，包含boot.s 和head.s 。boot.s 实现计算机加电自检引导扇区，第一次加载扇区和第二次加载操作系统的功能，head.s 主要包括初始设置的代码、时钟中断int 0x08的过程代码、系统调用中断int 0x80的过程代码以及任务A 和任务B 等的代码和数据。

（其中.S为扩展名的文件为汇编文件，为AT&T的386汇编语言，与intel格式只是语法不同，在AT&T中寄存器前冠以“%”而立即数前冠以“$”，十六进制立即数前冠以“0x”，在AT&T中第一个数是源操作数第二个数是目的操作数，。在Intel的语法中基寄存器用“[]”括起来而在AT&T中用“()”括起来,具体比较见：http://www.doc88.com/p-891572866672.html）

第二个文件夹fs ，也就是文件系统file system，里面的源程序关于打开文件，关闭文件，管道，读写，获取文件信息，释放块，串行io 等等。

第三个文件夹include ，这个里面的东西很复杂，各大部件（内存、io 、文件系统、进程）的头文件都聚集在这里。

第四个文件夹init ，里面只有一个main 函数。该函数的大致作用如下：

1．如何分配使用系统物理内存。

2．调用内核各部分的初始化函数。

3．Main 函数手工把自己移动到任务0中去运行（此时此时cpu 运行模式进入ring3）。

4．任务0创建一个进程（也就是进程1），来运行init 函数。任务0从此以后就是空转 的状态。

5．进程1运行init 函数。Init 的作用就是创建一个新的进程，运行shell 程序，当shell退出后，再重新创建。

Main 函数执行之后就循环等待，操作全交给了用户。

第五个文件夹kernel ，是内核代码的文件了，里面包括了进程创建fork.c ，内存分配malloc.c 等等内核函数，还包含有很多系统调用功能的实现函数的sys.c 等。

第六个文件夹lib ，库文件。

第七个文件夹mm ，memory manage内存管理，里面就一个文件memory .c。

2.linux启动过程分析：

来源：https://www.cnblogs.com/bluestorm/p/5981435.html

详细见下图

其中BIOS、BootLoader、uboot对比BIOS、BootLoader、uboot对比：

参见：http://blog.csdn.net/conowen/article/details/7253620

启动第一步－－加载BIOS

当你打开计算机电源，计算机会首先加载BIOS信息，BIOS信息是如此的重要，以至于计算机必须在最开始就找到它。这是因为BIOS中包含了CPU的相关信息、设备启动顺序信息、硬盘信息、内存信息、时钟信息、PnP特性等等。在此之后，计算机心里就有谱了，知道应该去读取哪个硬件设备了。

启动第二步－－读取MBR

众所周知，硬盘上第0磁道第一个扇区被称为MBR，也就是Master Boot Record，即主引导记录，它的大小是512字节，别看地方不大，可里面却存放了预启动信息、分区表信息。

系统找到BIOS所指定的硬盘的MBR后，就会将其复制到0x7c00地址所在的物理内存中。其实被复制到物理内存的内容就是Boot Loader，而具体到你的电脑，那就是lilo或者grub了。（其中后面源代码boot.s将被BIOS启动例程自动加载到内存的0x07c00处）

启动第三步－－Boot Loader

Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核做好一切准备。

Boot Loader有若干种，其中Grub、Lilo和spfdisk是常见的Loader。

我们以Grub为例来讲解吧，毕竟用lilo和spfdisk的人并不多。

系统读取内存中的grub配置信息（一般为menu.lst或grub.lst），并依照此配置信息来启动不同的操作系统。

启动第四步－－加载内核

根据grub设定的内核映像所在路径，系统读取内存映像，并进行解压缩操作。此时，屏幕一般会输出“Uncompressing Linux”的提示。当解压缩内核完成后，屏幕输出“OK, booting the kernel”。

系统将解压后的内核放置在内存之中，并调用start\_kernel()函数来启动一系列的初始化函数并初始化各种设备，完成Linux核心环境的建立。至此，Linux内核已经建立起来了，基于Linux的程序应该可以正常运行了。

启动第五步－－用户层init依据inittab文件来设定运行等级

内核被加载后，第一个运行的程序便是/sbin/init，该文件会读取/etc/inittab文件，并依据此文件来进行初始化工作。

其实/etc/inittab文件最主要的作用就是设定Linux的运行等级，其设定形式是“：id:5:initdefault:”，这就表明Linux需要运行在等级5上。Linux的运行等级设定如下：

0：关机

1：单用户模式

2：无网络支持的多用户模式

3：有网络支持的多用户模式

4：保留，未使用

5：有网络支持有X-Window支持的多用户模式

6：重新引导系统，即重启

关于/etc/inittab文件的学问，其实还有很多，在后序文章中设计到的，卖个关子，敬请期待，呵呵

启动第六步－－init进程执行rc.sysinit

在设定了运行等级后，Linux系统执行的第一个用户层文件就是/etc/rc.d/rc.sysinit脚本程序，它做的工作非常多，包括设定PATH、设定网络配置（/etc/sysconfig/network）、启动swap分区、设定/proc等等。如果你有兴趣，可以到/etc/rc.d中查看一下rc.sysinit文件，里面的脚本够你看几天的:P

启动第七步－－启动内核模块

具体是依据/etc/modules.conf文件或/etc/modules.d目录下的文件来装载内核模块。

启动第八步－－执行不同运行级别的脚本程序

根据运行级别的不同，系统会运行rc0.d到rc6.d中的相应的脚本程序，来完成相应的初始化工作和启动相应的服务。

启动第九步－－执行/etc/rc.d/rc.local

你如果打开了此文件，里面有一句话，读过之后，你就会对此命令的作用一目了然：

This script will be executed after all the other init scripts.

You can put your own initialization stuff in here if you don’t

want to do the full Sys V style init stuff.

rc.local就是在一切初始化工作后，Linux留给用户进行个性化的地方。你可以把你想设置和启动的东西放到这里。

启动第十步－－执行/bin/login程序，进入登录状态

此时，系统已经进入到了等待用户输入username和password的时候了，你已经可以用自己的帐号登入系统了。

具体（参考：http://blog.sina.com.cn/s/blog\_607787d30102w7lr.html

和linux 0.01内核分析和操作系统设计.pdf）

2.1 boot文件夹

Linux-0.01 的引导部分主要由两个源代码完成：boot.s 与 head.s 。boot.s 由BIOS 加载执行，head.s 是 32 位的引导代码，在最后会调用 main() 函数，完成系统的引导。

boot.s：BIOS 启动的时候加载并执行的代码；head.s：32 bit 的引导代码，调用 init\_main()。

head.s 由 boot.s 调用执行，当 head.s 执行时，系统运行在 32 位保护模式下。当 head.s 执行时，中断向量表与全局描述符表都已经被正确设置，并且合适的值被装入到 CPU 的帧、栈、堆栈指针寄存器中，然后检查有没有浮点数处理单元，如果没有，就在中断向量表中设置一个软件异常处理程序，便于模式浮点数运算。因为物理地址 0x00000000 实际上是页表数据的存放地址，因此在系统启动的最后阶段，所有的启动代码都被页表数据覆盖，启动代码将执行一个 jmp 指令，跳转到页表后的第一个地址，这个地址就是 \_main() 函数的入口地址，这时，系统就执行 init/main.c 中。

具体分析见源代码。

init文件夹：

就一个文件main.c。

在 init 目录下只有一个文件，main.c 。main.c 实现了系统的初始化功能，代码使用ANSI C写的，由 boot.s 调用。 main.c 根据机器 CMOS 的值来初始化系统时钟，然后启动 tty 设备，启动系统陷阱，启动进程调度器，启动文件系统，启动硬盘中断处理程序。接下来，内核开启中断，切换到用户模式下执行，最后，调用 init() 函数，进入进程调度循环。init() 函数会调用 setuup() 函数，setup() 函数会检查硬盘分区表，并装载磁盘分区。接下来，init() 函数 fork() 另外一个进程建立一个会话，然后使用 execve() 建立一个登录 shell 进程，这个 shell 进程的 HOME 被设置为 /usr/root 。在 main 不能执行函数调用，直到 调用 fork() 为止。