24.启动过程与系统管理

CentOS6.x的启动过程和相关配置文件

Linux系统的开机过程是比较复杂的一个过程，但整体上可以分成4大步骤或4大块：

1、按电源按钮后BIOS开始工作，然后进行加电自检

BIOS是英文"Basic Input Output System"的缩略词，就是"基本输入输出系统"。BIOS是电脑启动时加载的第一个软件。其实，它是一组固化到计算机内主板上一个ROM芯片上的程序。

2、MBR引导（Boot Loader）

主引导记录（MBR，Main Boot Record）是位于磁盘最前边的一段引导（Loader）代码。它负责磁盘操作系统(DOS)对磁盘进行读写时分区合法性的判别、分区引导信息的定位，它由磁盘操作系统(DOS)在对硬盘进行初始化时产生的。

3、启动内核

4、启动第一个进程init

具体的启动过程分析：

一、BIOS/开机自检

1.1 微控制器

系统想要启动必须先加载BIOS，按下电源键时，给微控制器下达一条复位指令，各寄存器复位，最后下达一条跳转指令，跳转到BIOS的ROM，使得硬件去读取主板上的BIOS程序，在这之前都是由硬件来完成，之后硬件就会把控制权交给BIOS；

1.2 BIOS->POST

随后BIOS程序加载CMOS（可读写的RAM芯片，保存BIOS设置硬件参数的数据）的信息，借CMOS取得主机的各项硬件配置；

取得硬件配置的信息之后，BIOS进行加电自检（Power-on self Test，POST）过程,检测计算机各种硬件信息，如果发现硬件错误则会报错（发出声音警告）；

之后BIOS对硬件进行初始化

BIOS将自己复制到物理内存中继续执行，开始按顺序搜寻可引导存储设备，判断的标准就是判断每个磁盘前512字节结尾是否存在55AA，有就是可引导，没有就继续检查下一个磁盘。一般第一张磁盘就是可引导磁盘，接下来就会读取磁盘的内容，但是要读取磁盘文件必须要有文件系统，这对BIOS挂载文件系统来说是不可能，因此需要一个不依赖文件系统的方法使得BIOS读取磁盘内容，这种方法就是引入MBR。最后

BIOS读取第一个可引导的存储设备的MBR（0柱面0磁道第一个扇区）中的boot loader。将MBR加载到物理内存中执行。

MBR载入内存后，BIOS将控制权转交给MBR（准确的说应该是MBR中的boot loader），然后MBR接管任务开始执行。

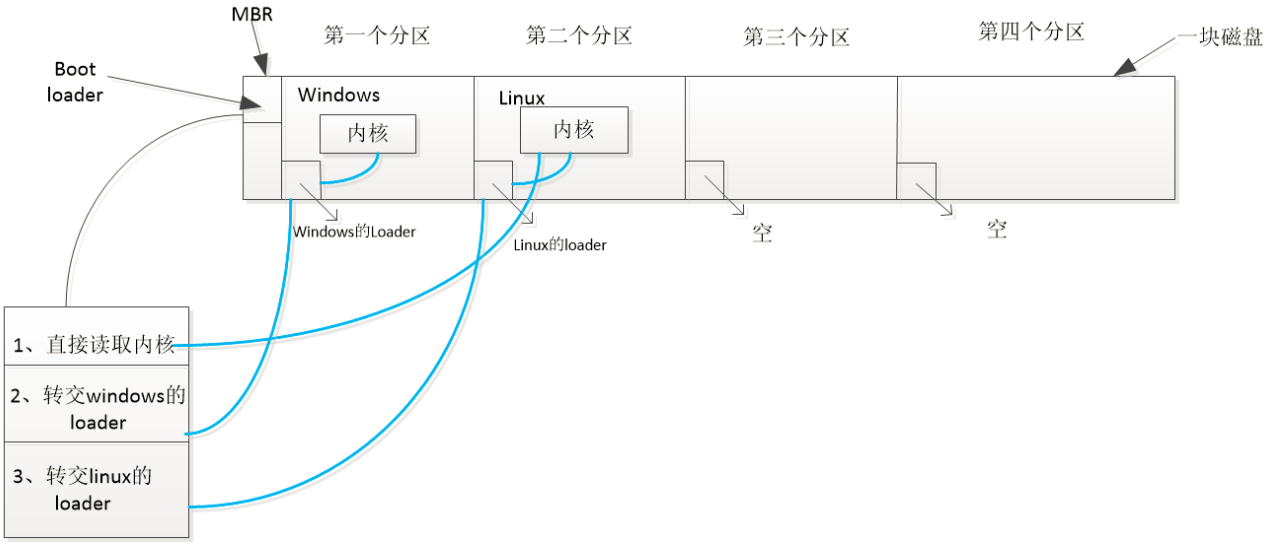
二、MBR引导（Boot Loader）

载入了第一个可引导的存储设备的MBR后，MBR中的boot loader就要读取所在磁盘的操作系统核心文件（即后面所说的内核）了。

2.1 boot loader

但是呢还存在一些问题，不同操作系统的文件系统格式不同？还有我们知道一个磁盘可以安装多个操作系统，boot loader怎么能够做到引导的就是我们想要的操作系统呢？这么多不同的功能单靠一个446字节的boot loader是远远不够的。因此必须弄一个相对应的程序来处理各自对应的操作系统核心文件，这个程序就是操作系统的loader（注意不是MBR中的boot loader），这样一来boot loader只需要将控制权交给对应操作系统的loader，让它负责去启动操作系统就行了。

这里有张图能更好地解释boot loader的作用：



解读上图内容，我们知道一个硬盘的每个分区的第一个扇区叫做**boot sector**，这个扇区存放的就是操作系统的**loader**，所以我们常说一个分区只能安装一个操作系统，如上图，第一个分区的boot sector存放着windows的loader，第二个分区放着Linux的loader，第三个第四个由于没有安装操作系统所以空着。至于MBR的boot loader是干嘛呢， boot loader有三个功能：**提供选单，读取内核文件，转交给其他loader**

提供选单就是给用户提供一张选项单，让用户选择进入哪个操作系统；

读取内核文件，我们知道系统会有一个默认启动的操作系统，这个操作系统的

loader在所在分区的boot sector有一份，除此之外，也会将这个默认启动的操作系统的loader复制一份到MBR的boot loader中，这样一来MBR就会直接读取boot loader中的loader了，然后就是启动默认的操作系统；

转交个其他的loader，当用户选择其他操作系统启动的时候，boot loader会将控制权转交给对应的loader，让它负责操作系统的启动。

另外，安装windows操作系统的时候，windows会主动复制一份自己的loader到MBR中的boot loader中，这种操作在linux下不会。所以我们安装多重操作系统的时候要求先安装windows，然后再安装Linux；我们假设先安装Linux，再安装windows的时候就会自动把windows的loader复制到MBR中的boot loader，这样一来就会默认优先启动windows。然而先安装windows，自动复制windows的loader到boot loader，再安装Linux的时候，我们可以设置把Linux的loader复制到boot loader中，把原先windows的覆盖掉，这样才能设置Linux默认启动。

2.2 Linux的GRUB

Linux操作系统层面的loader使用的是grub。

boot loader和grub到底是什么关系呢？

进入/boot/grub目录下：

```
[root@CentOS6 ~]# cd /boot/grub
[root@CentOS6 grub]# ll
总用量 274
-rw-r--r--. 1 root root     63 1月  27 04:23 device.map
-rw-r--r--. 1 root root  13428 1月  27 04:23 e2fs_stage1_5
-rw-r--r--. 1 root root  12636 1月  27 04:23 fat_stage1_5
-rw-r--r--. 1 root root  11780 1月  27 04:23 ffs_stage1_5
-rw-------. 1 root root    745 1月  27 04:23 grub.conf
-rw-r--r--. 1 root root  11772 1月  27 04:23 iso9660_stage1_5
-rw-r--r--. 1 root root  13284 1月  27 04:23 jfs_stage1_5
lrwxrwxrwx. 1 root root     11 1月  27 04:23 menu.lst -> ./grub.conf
-rw-r--r--. 1 root root  11972 1月  27 04:23 minix_stage1_5
-rw-r--r--. 1 root root  14428 1月  27 04:23 reiserfs_stage1_5
-rw-r--r--. 1 root root   1341 11月 15 2010 splash.xpm.gz
-rw-r--r--. 1 root root    512 1月  27 04:23 stage1
-rw-r--r--. 1 root root 126148 1月  27 04:23 stage2
-rw-r--r--. 1 root root  12040 1月  27 04:23 ufs2_stage1_5
-rw-r--r--. 1 root root  11380 1月  27 04:23 vstafs_stage1_5
-rw-r--r--. 1 root root  13980 1月  27 04:23 xfs_stage1_5
[root@CentOS6 grub]#
```

我们可以看到很多文件，其实Linux的loader为stage1那个文件，Linux所在分区的boot sector（一个扇区是512字节）就是存放着stage1文件的内容，同时默认Linux启动的话，也需要把stage1中的引导代码安装到MBR中的boot loader中。该文件太小，能完成的功能有限，因此Linux的loader只是简单的引导作用。

stage1完成了主程序的引导后，主引导程序开始加载配置文件了，但是加载这些配置文件之前需要有文件系统的支持，可是现在还没有文件系统呢，**grub在不依赖Linux内核的情况下具有读取配置文件与内核映像的能力**。grub的内置文件系统其

实是依靠stage1_5这些文件定义的，而且有不同文件系统对应不同的stage1_5文件。

而后开始读取stage2开始真正地读取配置文件grub.conf。解析/boot/grub/grub.conf文件：

vim /boot/grub/grub.conf

default=0　设定默认启动菜单项，当系统中有多个内核时，0表示默认加载第1个，1表示第2个内核

timeout=5　菜单项等待选项时间为5s

splashimage=(hd0,0)/grub/splash.xpm.gz　指明菜单背景图片路径为

hiddenmenu　隐藏菜单

title CentOS 6 (2.6.32-358.6.1.el6.x86_64)　定义菜单项

root (hd0,0)　grub查找stage2及kernel文件所在设备分区，grub的根：/boot/

kernel /vmlinuz-2.6.32-642.el6.x86_64 ro root=UUID=8e411d32-58df-4366-bddf-02f51b7c0f6c rd_NO_LUKS KEYBOARDTYPE=pc KEYTABLE=us rd_NO_MD crashkernel=auto LANG=zh_CN.UTF-8 rd_NO_LVM rd_NO_DM rhgb quiet　启动的内核

initrd /initramfs-2.6.32-358.6.1.el6.x86_64.img　内核匹配的ramfs文件系统，虚拟文件系统（存在于内存中的文件系统）

总结：MBR最后最核心的作用就是加载内核文件的

三、grub把内核文件加载到内存后

3.1 加载内核文件

MBR将内核文件（代码）载入物理内存中执行，内核就是/boot/vmlinuz-2.6.32-696.el6.x86_64，观察该文件，发现这是一个压缩镜像文件。

```
[root@CentOS6 boot]# ll
总用量 33647
-rw-r--r--. 1 root root   108103 5月  11 2016 config-2.6.32-642.el6.x86_64
drwxr-xr-x. 3 root root     1024 1月  27 04:21 efi
drwxr-xr-x. 2 root root     1024 3月   1 23:22 grub
-rw-------. 1 root root 22131982 1月  27 04:23 initramfs-2.6.32-642.el6.x86_64.img
-rw-------. 1 root root  5092759 1月  27 04:35 initrd-2.6.32-642.el6.x86_64kdump.img
drwx------. 2 root root    12288 1月  27 04:14 lost+found
-rw-r--r--. 1 root root   215559 5月  11 2016 symvers-2.6.32-642.el6.x86_64.gz
-rw-r--r--. 1 root root  2615003 5月  11 2016 System.map-2.6.32-642.el6.x86_64
-rwxr-xr-x. 1 root root  4264528 5月  11 2016 vmlinuz-2.6.32-642.el6.x86_64
```

控制权转交给内核后，内核重新检测各种硬件信息，（第一次为POST自检）我们前边说了，一个完整的Linux包括内核和内核之上的程序要使用硬件，还需要加载提供这些程序功能的模块，然而这些模块都在根目录的/lib/modules/2.6.32-696.el6.x86_64下（/和/lib/modules/不能挂载不同的分区），这时候内核还没有

文件系统的概念，没有文件系统就没办法挂载根目录，想要挂载根目录就需要相应的模块支持，而我们原本的问题就是如何加载模块，先有鸡后有蛋。

3.2 加载initrd初始化ramfs文件系统

/boot/initramfs-2.6.32-696.el6.x86_64.img文件就是解决上面问题的，我们来看一下这个文件：

```
file initramfs-2.6.32-696.el6.x86_64.img # 查看该文件类型
cp initramfs-2.6.32-696.el6.x86_64.img /app
cd /app
mv initramfs-2.6.32-696.el6.x86_64.img initramfs-2.6.32-696.el6.x86_64.img.gz     # gzip解压文件必须以.gz后缀
gzip -d initramfs-2.6.32-696.el6.x86_64.img.gz
file initramfs-2.6.32-696.el6.x86_64.img       # 查看需要借助cpio命令
mkdir init
cd init
cpio -idv < /app/initramfs-2.6.32-696.el6.x86_64.img # 解压至/app/init目录下
```

```
[root@CentOS6 init]# ls
bin      dracut-004-409.el6  init              initqueue-settled  lib64    pre-mount    pre-udev  sys      usr
cmdline  emergency           initqueue         initqueue-timeout  mount    pre-pivot    proc      sysroot  var
dev      etc                 initqueue-finished  lib              netroot  pre-trigger  sbin      tmp
```

我们发现解压之后的内容类似于真正/目录下内容，这是因为这是一个最小化的Linux根文件系统。内核就是先把这个文件展开，**形成一个虚拟文件系统**，内核借虚拟文件系统装载必要的模块，**完成后释放该虚拟文件系统并挂载真正的根目录。**

四、启动第一个进程init

4.1 init进程：主要功能是准备软件执行的环境

内核完成硬件检测和加载模块后，内核会呼叫第一个进程，就是/sbin/init，至此内核把控制权交给init进程

读取初始化配置文件/etc/inittab，决定操作系统的runlevel，/etc/inittab内有这样一句：

id:runlevel:action:process ，　这里我的实验机器的值：

id:3:initdefault:

id	代表设定的项目，没有具体的实际意义
runlevel	执行级别，0-关机、1-单用户、2-没有NFS的多用户、3-真正的多用户、4-预留、5-Xwindows、6-reboot
action	init的动作行为，initdefault表示要默认启动的runlevel
process	执行动作的指令，一般为脚本文件

4.2 /etc/rc.d/rc.sysinit

读取/etc/rc.d/rc.sysinit系统初始化脚本，设置主机名，挂载/etc/fstab中的文件系统，修改/etc/sysctl.conf 的内核参数等各项系统环境。

查看该脚本内容，大致功能如下：

定义主机名，如果不存在则将主机名定义为localhost；

读取/etc/sysconfig/network文件，设置网络环境；

挂载内存装置/proc和/sys和USB装置，如果USB装置存在，则会加载usb模块并挂载usb文件系统；

接下来是SELINUX的一些相关设置；

设定text banner，显示欢迎界面；

...

将开机启动信息存放到/var/log/dmesg中。

4.3 /etc/rc.d/rc

执行/etc/rc.d/rc脚本，下面是/etc/rc.d/rc脚本中我们关心的代码部分：

```
# First, run the KILL scripts.
for i in /etc/rc$runlevel.d/K* ; do
# Check if the subsystem is already up.
subsys=${i#/etc/rc$runlevel.d/K??}
[ -f /var/lock/subsys/$subsys -o -f /var/lock/subsys/$subsys.init ] || continue
check_runlevel "$i" || continue
# Bring the subsystem down.
[ -n "$UPSTART" ] && initctl emit --quiet stopping JOB=$subsys
$i stop
[ -n "$UPSTART" ] && initctl emit --quiet stopped JOB=$subsys
done
# Now run the START scripts.
for i in /etc/rc$runlevel.d/S* ; do
# Check if the subsystem is already up.
subsys=${i#/etc/rc$runlevel.d/S??}
[ -f /var/lock/subsys/$subsys ] && continue
[ -f /var/lock/subsys/$subsys.init ] && continue
check_runlevel "$i" || continue
```

根据运行级别（0123456）进入相应的/etc/rc.d/rcN.d目录，启动和关闭相关的系统服务。里边存放着一堆以K和S开头的软链接文件，分别代表对应的服务。K开头表示该运行级别下需要把该服务杀死，S开头表示该运行级别下需要把该服务开

启。，上诉操作都是由/etc/rc.d/rc脚本来完成的。另外我们还注意都S和K后边的数字，他们的数字代表了读取的顺序，因为有些服务是具有一定的关联性。

而且每个rcN.d目录内最后都会有一个S99local文件 ，该文件指向../rc.local脚本。

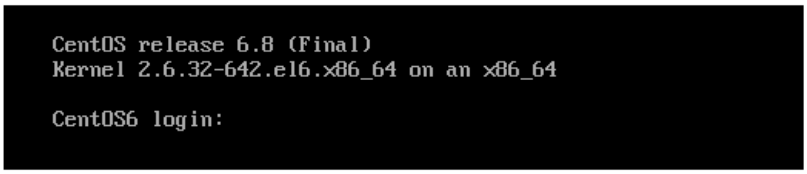
4.4 /etc/rc.d/rc.local

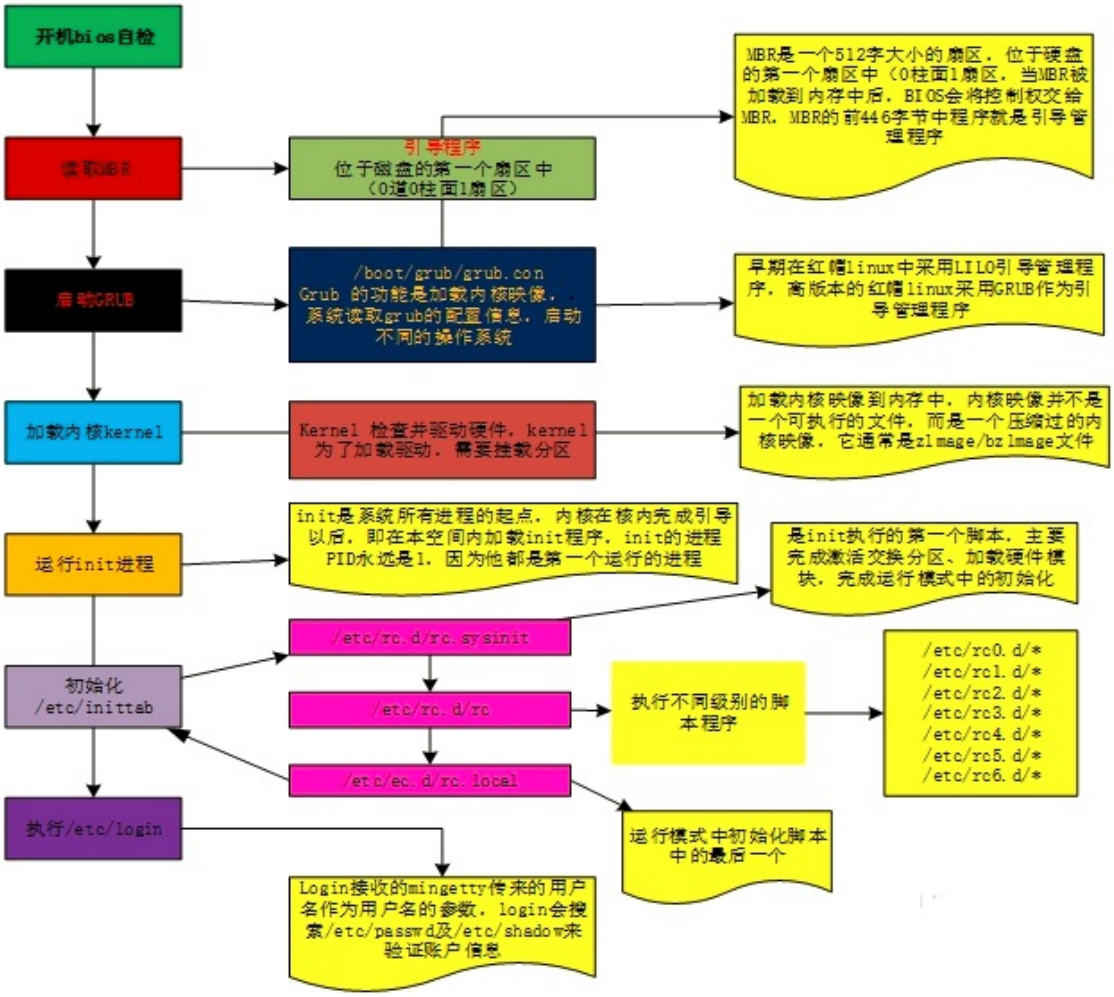
系统根据runlevel执行完/etc/rc.d/rcN.d中的脚本后，调用/etc/rc.d/rc.local脚本

这时候系统已经完成了各种必要系统服务的启动，**假如我们想自定义一些指令要在开机的时候启动，我们就可以把他们放到/etc/rc.d/rc.local内，该文件默认为空。**

4.5 启动终端

接下来会由/sbin/mingetty指令启动终端，由于系统设置启动tty1-tty6 ，所以会启动6个命令行终端。最终呈现给我们的就是这样一个画面：





CentOS7.x的启动过程和相关配置文件

CentOS7和CentOS6启动流程差不多，只不过到init程序时候，改为了systemd，因此详细解释一下systemd后的启动流程：

1. uefi或BIOS初始化，开始post开机自检
2. 加载mbr到内存
3. GRUB阶段，CentOS6加载的是/boot/grub/grub.conf ， 而CentOS7加载的是/boot/grub2/grub.cfg
4. 加载内核和initramfs模块
5. 内核开始初始化，使用systemd来代替centos6以前的init程序,systemd使用“target”来处理引导和服务管理过程。这些systemd里的“target”文件被用于分组不同的引导单元以及启动同步进程。

systemd执行的第一个目标是/etc/systemd/system/default.target。但实际上default.target是指向/usr/lib/systemd/system/multi-user.target的软链接。会启动如下两个目录中单元。

/etc/systemd/system/multi-user.target.wants/

/usr/lib/systemd/system/multi-user.target.wants/

multi-user.targe 文件内容如下：

[Unit]

Description=Multi-User System

Documentation=man:systemd.special(7)

Requires=basic.target

Conflicts=rescue.service rescue.target

After=basic.target rescue.service rescue.target

AllowIsolate=yes

在这个阶段，这个target的执行为多用户支持设定系统环境。非root用户会在这个阶段的引导过程中启用。防火墙相关的服务也会在这个阶段启动。**"multi-user.target"执行完后，会将控制权交给另一层“basic.target”。**它通过如下两个目录决定那些单元会被启动。

/etc/systemd/system/basic.target.wants/

/usr/lib/systemd/system/basic.target.wants/

basi.target 文件内容如下：

[Unit]

Description=Basic System

Documentation=man:systemd.special(7)

Requires=sysinit.target

After=sysinit.target

Wants=sockets.target timers.target paths.target slices.target

After=sockets.target paths.target slices.target

"basic.target"单元用于启动普通服务。

basic.target之后将控制权交给**sysinit.target.** **这个target有点像CentOS6钟的rc.sysinit脚本，**它通过如下两个目录决定那些单元会被启动。

/etc/systemd/system/sysinit.target.wants/

/usr/lib/systemd/system/sysinit.target.wants/

sysinit.target文件内容如下：

[Unit]

Description=System Initialization

Documentation=man:systemd.special(7)

Conflicts=emergency.service emergency.target

Wants=local-fs.target swap.target

After=local-fs.target swap.target emergency.service emergency.target

sysinit.target会启动重要的系统服务例如系统挂载，内存交换空间和设备，内核补充选项等等。**sysinit.target在启动过程中会传递给 local-fs.target 和 swap.target。**它通过如下一个目录决定那些单元会被启动。

/usr/lib/systemd/system/local-fs.target.wants/

local-fs.target 文件内容如下：

[Unit]

Description=Local File Systems

Documentation=man:systemd.special(7)

DefaultDependencies=no

Conflicts=shutdown.target

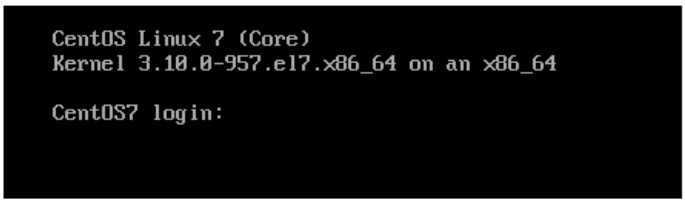
After=local-fs-pre.target

OnFailure=emergency.target

OnFailureJobMode=replace-irreversibly

local-fs.target，这个target单元不会启动用户相关的服务，它只处理底层核心服务。这个target会根据/etc/fstab来执行相关操作。**原来的/etc/inittab,/etc/rc.d/rc.locale这些配置文件，就不在启动过程中生效！**

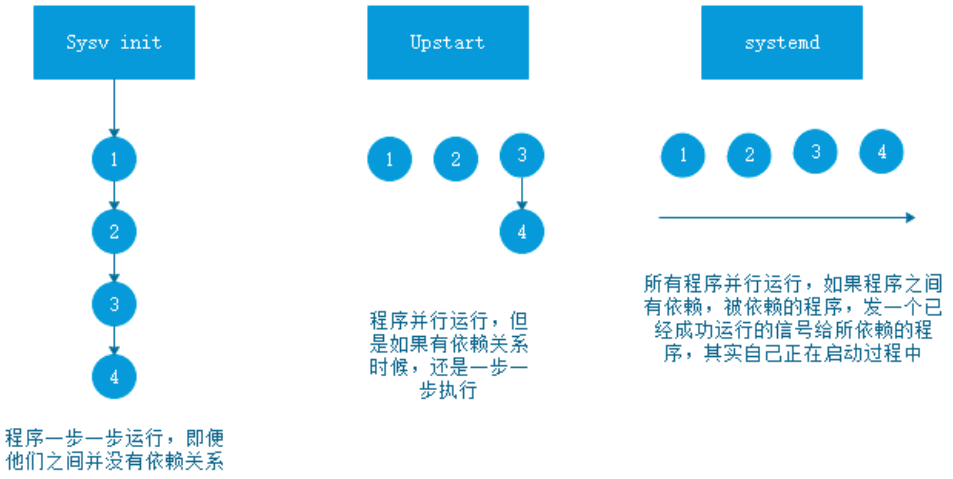
6.最后执行getty.target及登录服务



7.安装了图形界面的话，在执行graphical.target需要的服务

CentOS5，6，7启动区别

系统启动和服务器守护进程管理器，它不同于centos5的Sysv init，centos6的Upstart（Ubuntu制作出来），systemd是由Redhat的一个员工首先提出来的，它在内核启动后，服务什么的全都被systemd接管，kernel只是用来管理硬件资源，相当于内核被架空了，因此linus很不满意Redhat这种做法。



Systemd相关的知识点

系统启动和服务器守护进程管理器，负责在系统启动或运行时，激活系统资源，服务器进程和其它进程。

核心概念：`unit`

`unit`表示不同类型的systemd对象，通过配置文件进行标识和配置；文件中主要包含了系统服务、监听socket、保存的系统快照以及其它与init相关的信息

配置文件：

- `/usr/lib/systemd/system`:每个服务最主要的启动脚本设置，类似于之前的/etc/init.d/
- `/run/systemd/system`：系统执行过程中所产生的服务脚本，比上面目录优先运行

- `/etc/systemd/system`：管理员建立的执行脚本，类似于/etc/rc.d/rcN.d/Sxx类的功能，比上面目录优先运行

unit类型，# systemctl -t help

- **`Service unit`**：文件扩展名为.service，用于定义系统服务
- **`Target unit`**：文件扩展名为.target，用于模拟实现运行级别
- **`Device unit`**：.device，用于定义内核识别的设备
- **`Mount unit`**：.mount，定义文件系统挂载点
- **`Socket unit`**：.socket，用于标识进程间通信用的socket文件，也可在系统启动时，延迟启动服务，实现按需启动
- **`Snapshot unit`**：.snapshot，管理系统快照
- **`Swap unit`**：.swap，用于标识swap设备
- **`Automount unit`**：.automount，文件系统的自动挂载点
- **`Path unit`**：.path，用于定义文件系统中的一个文件或目录使用,常用于当文件系统变化时，延迟激活服务，如：spool 目录

service unit文件格式

在unit文件中，以“#” 开头的行后面的内容会被认为是注释，相关布尔值，1、yes、on、true 都是开启，0、no、off、false 都是关闭，时间单位默认是秒，所以要用毫秒（ms）分钟（m）等须显式说明

service unit file文件通常由三部分组成

- [Unit]：定义与Unit类型无关的通用选项；用于提供unit的描述信息、unit行为及依赖关系等
- [Service]：与特定类型相关的专用选项；此处为Service类型
- [Install]：定义由“systemctl enable”以及"systemctl disable“命令在实现服务启用或禁用时用到的一些选项

unit段的常用选项

- Description：描述信息
- After：定义unit的启动次序，表示当前unit应该晚于哪些unit启动，其功能与Before相反
- Requires：依赖到的其它units，强依赖，被依赖的units无法激活时，当前unit也无法激活
- Wants：依赖到的其它units，弱依赖
- Conflicts：定义units间的冲突关系

Service段的常用选项

- PIDFile：值是指定一个文件，这个文件保存的是当前服务启动起来后的PID
- EnvironmentFile：环境配置文件
- ExecStart：指明启动unit要运行命令或脚本的绝对路径

- ExecStartPre：ExecStart前运行
- ExecStartPost：ExecStart后运行
- ExecStop：指明停止unit要运行的命令或脚本
- Restart：当设定Restart=1 时，则当次daemon服务意外终止后，会再次自动启动此服务
- Type：定义影响ExecStart及相关参数的功能的unit进程启动类型
    - **simple：默认值，这个daemon主要由ExecStart接的指令串来启动，启动后常驻于内存中**
    - **forking：由ExecStart启动的程序做父进程延申出其他子程序来作为此daemon的主要服务。原生父程序在启动结束后就会终止**
    - oneshot：与simple类似，不过这个程序在工作完毕后就结束了，不会常驻在内存中
    - dbus：与simple类似，但这个daemon必须要在取得一个D-Bus的名称后，才会继续运作.因此通常也要同时设定BusNname= 才行，和socket编程有关系
    - notify：在启动完成后会发送一个通知消息。
    - idle：与simple类似，要执行这个daemon必须要所有的工作都顺利执行完毕后才会执行。这类的daemon通常是开机到最后才执行即可的服务

Install段的常用选项

- Alias：别名，可使用systemctl command Alias.service
- RequiredBy：被哪些units所依赖，强依赖
- WantedBy：被哪些units所依赖，弱依赖，一般情况下就是指定这个服务在那个target目标环境下运行，比如multi-user.target
- Also：安装本服务的时候还要安装别的相关服务

注意：

对于新创建的unit文件，或者修改了的unit文件，要通知systemd重载此配置文件,而后可以选择重启

```
# systemctl  daemon-reload
```

Systemd管理服务

服务启动，重启方面：

systemctl COMMAND name.service

-	centOS6	CentOS7
启动	service name start	systemctl start name.service
停止	service name stop	systemctl stop name.service
重启	service name restart	systemctl restart name.service
状态	service name status	systemctl status name.service
条件式重启(已启动才重启，否则不做操作)	service name condrestart	systemctl try-restart name.service
重载或重启服务(先加载，再启动)	-	systemctl reload-or-restart name.service
重载或条件式重启服务	-	systemctl reload-or-try-restart name.service
禁止自动和手动启动	-	systemctl mask name.service
取消禁止	-	systemctl unmask name.service

服务查看

查看所有服务	-	systemctl list-units --type service --all

chkconfig命令的对应关系

-	centOS6	CentOS7
设定某服务开机自启	chkconfig name on	systemctl enable name.service
设定某服务开机禁止启动	chkconfig name off	systemctl disable name.service
查看所有服务的开机自启状态	chkconfig --list	systemctl list-unit-files --type service
用来列出该服务在哪些运行级别下启用和禁用	chkconfig sshd --list	ls /etc/systemd/system/*.wants/sshd.service
查看服务是否开机自启	-	systemctl is-enabled name.service

其他命令

-	centOS6	CentOS7
查看服务的依赖关系	-	systemctl list-dependencies name.service
杀掉进程	-	systemctl kill unitname
切换至紧急救援模式	-	systemctl rescue
切换至emergency模式	-	systemctl emergency
关机	-	systemctlhalt、systemctlpoweroff
重启	-	systemctl reboot
挂起	-	systemctl suspend
休眠	-	systemctl hibernate
休眠并挂起	-	systemctlhybrid-sleep

服务状态，systemctl list-unit-files --type service --all #显示状态

- loaded:Unit配置文件已处理

- active(running):一次或多次持续处理的运行
- active(exited):成功完成一次性的配置
- active(waiting):运行中，等待一个事件
- inactive:不运行
- enabled:开机启动
- disabled:开机不启动
- static:开机不启动，但可被另一个启用的服务激活

实例，编写一个service unit来启动脚本把systemd相关的知识点串一串，用一下

在来一个实例：CentOS7的新特性timer，也是交给systemd服务来管理，systemctl命令

timer单元的常用选项：[Service]-----》[Timer]

单调定时器

```
OnActiveSec=        表示相对于本单元被启用的时间点
OnBootSec=          表示相对于机器被启动的时间点
OnStartupSec=       表示相对于systemd被首次启动的时间点
OnUnitActiveSec=    表示相对于匹配单元(本标签下Unit=指定的单元)最后一次被启动的时间点
OnUnitInactiveSec= 表示相对于匹配单元(本标签下Unit=指定的单元)最后一次被停止的时间点
```

日历定时器OnCalendar=　　crontab ：　* * * * * 命令

```
Thu,Fri 2022-*-1,5 11:12:13  #表示2022年任意月份的1日和5日，如果是星期四或星期五，则在时间11:12:13执行
*-*-* *:*:00                 #表示每分钟
*-*-* 00:00:00               #表示每天
*-01,07-01 00:00:00          #表示每半年
*:0/15                       #表示每15分钟
12,14,13:20,10,30            #表示12/13/14点的10分、20分、30分
Mon,Fri *-01/2-01,03 *:30:45 #表示任意年份奇数月份的1日和3日，如果是周一或周五，则在每小时的30分45秒执行
```

Unit=要执行的单元

CentOS7上的运行级别

在centOS7上运行级别的含义已经和之前不同了，CentOS6的运行级别就是通过开启关闭不同的服务产生的效果，在从centOS7上，已然由.target来代替运行级别，我们可以称target为目标态，我们可以通过target定制更符合我们工作运行环境。

我们可以通过命令：`ls /usr/lib/systemd/system/*.target`查看我们的机器上有多少个target，有59个之多

```
[root@CentOS7 system]# ls /usr/lib/systemd/system/*.target
/usr/lib/systemd/system/basic.target                /usr/lib/systemd/system/poweroff.target
/usr/lib/systemd/system/bluetooth.target            /usr/lib/systemd/system/printer.target
/usr/lib/systemd/system/cryptsetup-pre.target       /usr/lib/systemd/system/reboot.target
/usr/lib/systemd/system/cryptsetup.target           /usr/lib/systemd/system/remote-cryptsetup.target
/usr/lib/systemd/system/ctrl-alt-del.target         /usr/lib/systemd/system/remote-fs-pre.target
/usr/lib/systemd/system/default.target              /usr/lib/systemd/system/remote-fs.target
/usr/lib/systemd/system/emergency.target            /usr/lib/systemd/system/rescue.target
/usr/lib/systemd/system/final.target                /usr/lib/systemd/system/rpcbind.target
/usr/lib/systemd/system/getty-pre.target            /usr/lib/systemd/system/runlevel0.target
/usr/lib/systemd/system/getty.target                /usr/lib/systemd/system/runlevel1.target
/usr/lib/systemd/system/graphical.target            /usr/lib/systemd/system/runlevel2.target
/usr/lib/systemd/system/halt.target                 /usr/lib/systemd/system/runlevel3.target
/usr/lib/systemd/system/hibernate.target            /usr/lib/systemd/system/runlevel4.target
/usr/lib/systemd/system/hybrid-sleep.target         /usr/lib/systemd/system/runlevel5.target
/usr/lib/systemd/system/initrd-fs.target            /usr/lib/systemd/system/runlevel6.target
```

使用`systemctl list-unit-files --type target --all`可以查看所有目标态的状态，或者`systemctl list-dependencies runleve3.target`命令查看目标态的依赖性。

```
[root@CentOS7 system]# systemctl list-dependencies runlevel3.target
runlevel3.target
● ├─abrt-ccpp.service
● ├─abrt-oops.service
● ├─abrt-vmcore.service
● ├─abrt-xorg.service
● ├─abrtd.service
● ├─auditd.service
● ├─chronyd.service
● ├─crond.service
● ├─dbus.service
● ├─irqbalance.service
● ├─network.service
● ├─NetworkManager.service
● ├─plymouth-quit-wait.service
● ├─plymouth-quit.service
● ├─postfix.service
● ├─rhel-configure.service
● ├─rpcbind.service
● ├─rsyslog.service
```

在centOS7上所谓的目标态，其实就是由各种指定的服务和基础target组合而成的。

CentOS7 grub2

在centOS6上，我们的grub文件是`/boot/grub/grub.conf` ，这个要修改直接用vi来修改，但是CentOS7不在是直接修改了！

在centOS7上，文件改成`/boot/grub2/grub.cfg`了，但是功能还是大致一样的都是用于加载内核的，不过在centOS7上设置默认启动项发生了一些变化，假如我们现在有两个内核，我们需要改变默认启动应该如何做到呢？

首先，`vim /etc/default/grub`打开如下文件：

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
```

GRUB_TERMINAL_OUTPUT="console"

GRUB_CMDLINE_LINUX="rhgb quiet"

GRUB_DISABLE_RECOVERY="true"

打开文件后，我们修改GRUB_DEFAULT的值，和centOS6一样，0代表第一个内核，1代表第二个，以此类推。

我们在修改完成后，并没有立即生效，使用**grub2-mkconfig -o /boot/grub2/grub.cfg**命令来生成grub2.cfg文件，我们在下次启动的时候就会默认选择新的默认内核。

启动引导程序Grub的加密

前面给大家讲过，

基于centos6进行grub加密

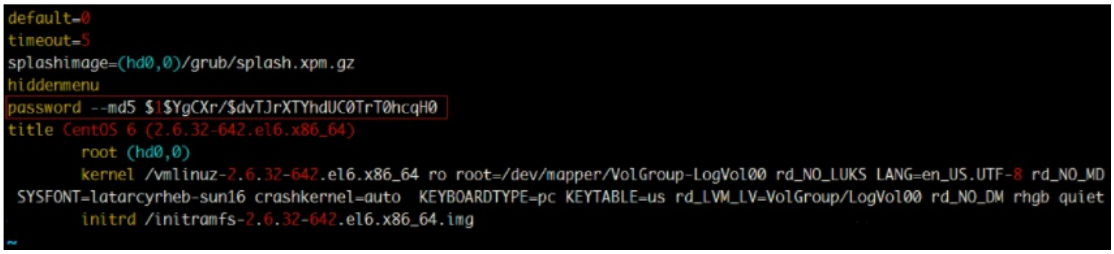
1.Linux系统中grub.conf 配置文件可为为GRUB菜单设置密码保护，在/boot/grub/grub.conf或/etc/grub.conf文件中的title字段上面新增一行。

A. password --md5 PASSWD（md5方式加密）

B. password PASSWRD（明文密码加密）

2.使用md5加密方式加密，md5加密口令的生成命令为grub-md5-crypt 如图：

3..复制第2步中红框处内的字符串，打开grub配置文件vim /boot/grub/grub.conf ，添加如下一行，如图：



4.添加完成后wq保存退出，重启系统reboot，此时再想编辑grub菜单，系统就会提示按p键，验证密码才可以进行编辑。

基于centos7进行grub加密

 # grub2-mkpasswd-pbkdf2

输入口令： 123456

Reenter password: 123456

PBKDF2 hash of your password is

grub.pbkdf2.sha512.10000.8F355BAB512AFB7B8C990A1FEB887B8F2F3F1C54467E9B9F0535F2268E1FFC5F4E8D33F76....

 # vim /etc/grub.d/00_header #在最后后面添加如下内容

cat <<EOF

set superusers='root'

password_pbkdf2 root

grub.pbkdf2.sha512.10000.8F355BAB512AFB7B8C990A1FEB887B8F2F3F1C54467E9B9F0535F2268E1FFC5F4E8D33F7633

D7FBEC25B2039C6D8B3226A90528D4883AB9B99E391A4965D069F.DDE992693BE2C09FFEEC1149120B6B84DBAB933DE6CF7BFF718

E1DDC858AB73EE32C

FF45EB7F06AC45AA6792E91C4CD09E2B445FC288C47E79F537DBBABAD756

EOF

```
cat <<EOF
set superusers='root'
password_pbkdf2 root grub.pbkdf2.sha512.10000.8F355BAB512AFB7B8C990A1FEB887B8F2F3F1C54467E9B9F0535F2268E1FFC5F4E8D33F7633
D7FBEC25B2039C6D8B3226A90528D4883AB9B99E391A4965D069F.DDE992693BE2C09FFEEC1149120B6B84DBAB933DE6CF7BFF718E1DDC858AB73EE32C
FF45EB7F06AC45AA6792E91C4CD09E2B445FC288C47E79F537DBBABAD756
EOF
```

重新编译生成grub.cfg文件

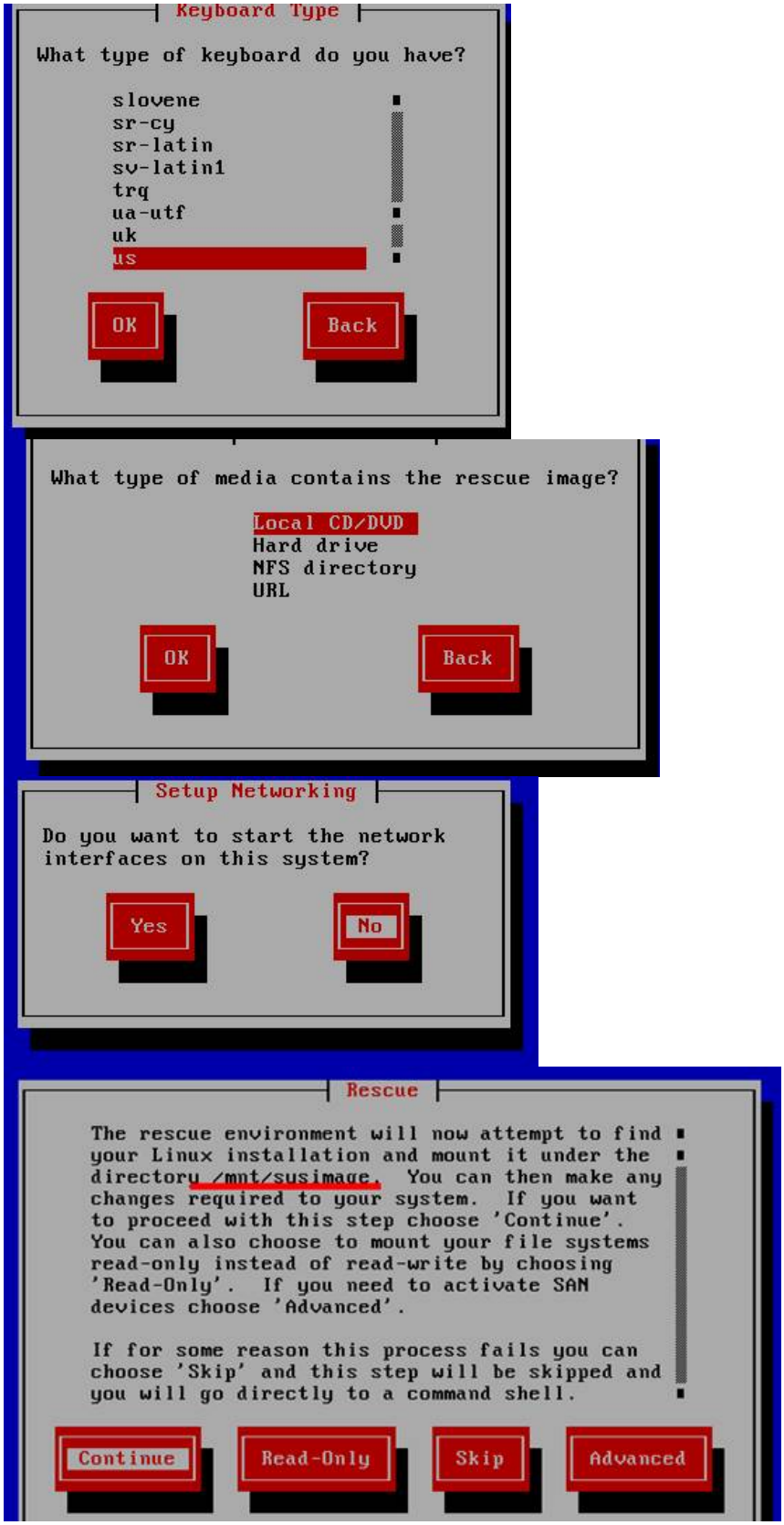
```
grub2-mkconfig  -o  /boot/grub2/grub.cfg
```

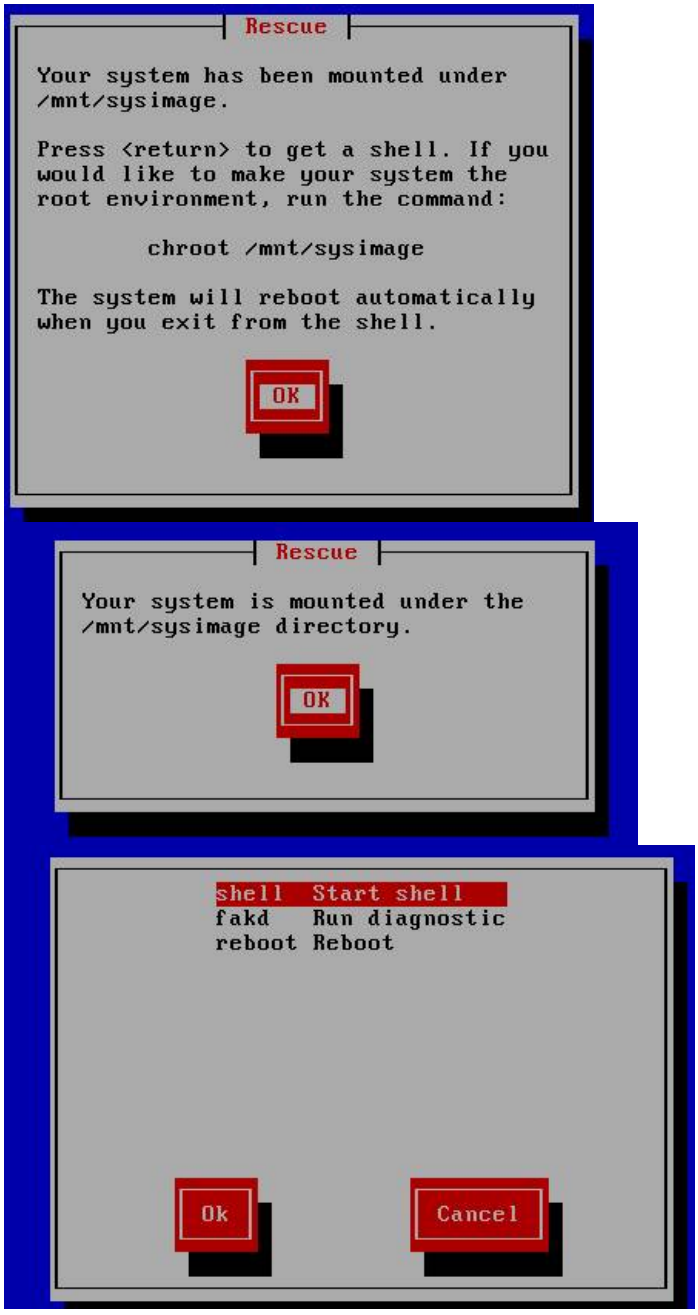
重启搞定

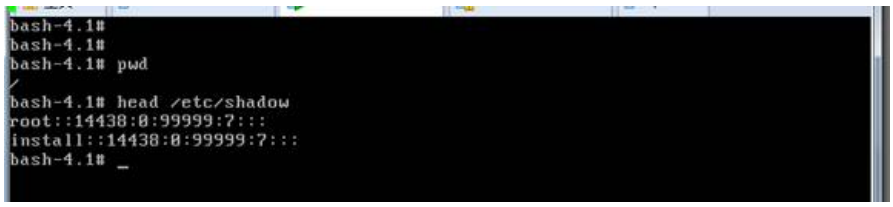
系统修复

基于6版本系统进入救援模式

修改BIOS启动顺序，直接以光盘引导系统

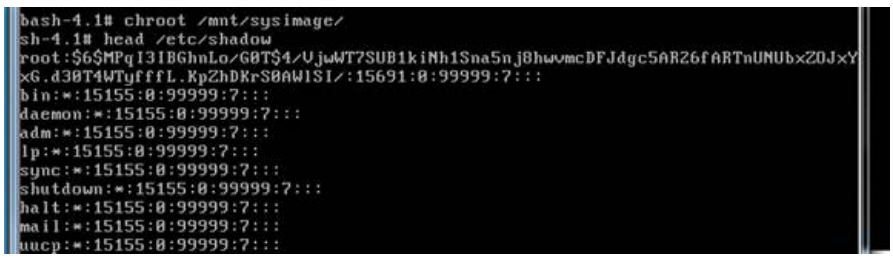

Keyboard Type
What type of keyboard do you have?
slovene
sr-cy
sr-latin
sv-latin1
trq
ua-utf
uk
us
OK
Back
What type of media contains the rescue image?
Local CD/DVD
Hard drive
NFS directory
URL
OK
Back
Setup Networking
Do you want to start the network interfaces on this system?
Yes
No
Rescue
The rescue environment will now attempt to find your Linux installation and mount it under the directory /mnt/sysimage. You can then make any changes required to your system. If you want to proceed with this step choose 'Continue'. You can also choose to mount your file systems read-only instead of read-write by choosing 'Read-Only'. If you need to activate SAN devices choose 'Advanced'.
If for some reason this process fails you can choose 'Skip' and this step will be skipped and you will go directly to a command shell.
Continue
Read-Only
Skip
Advanced





ramfs ： 内存文件系统

chroot /mnt/sysimage # 切换文件系统根



实验：在centOS7下破坏前446字节并修复

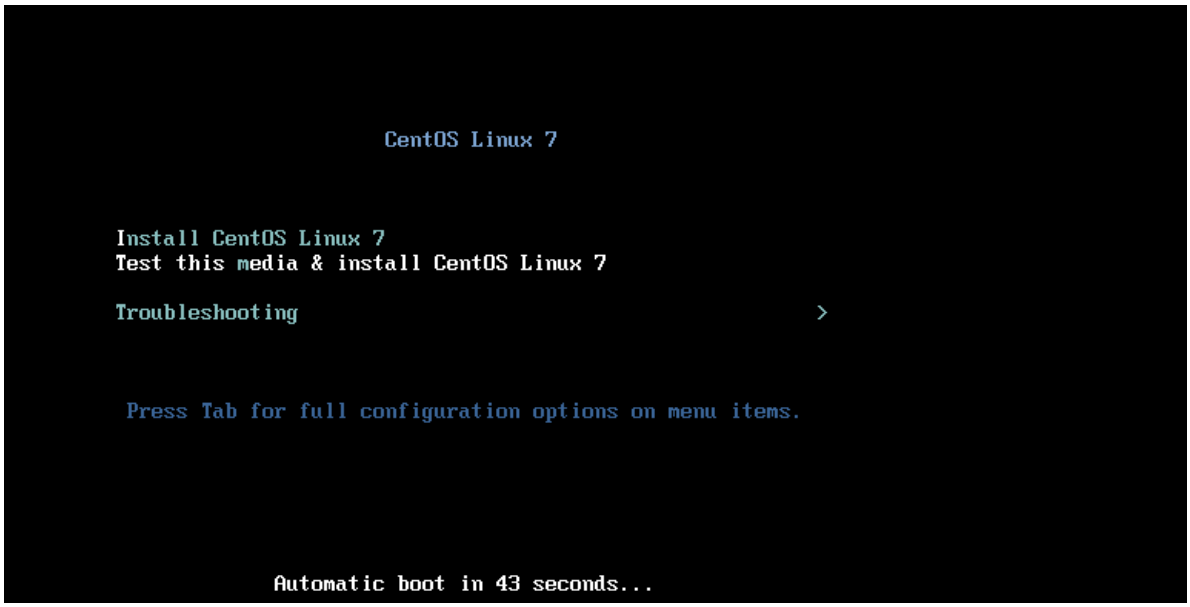
第一步：破坏硬盘的前446字节：

```
# dd if=/dev/zero of=/dev/sda bs=1 count=446
```

```
# hexdump -C -n 512 /dev/sda
```

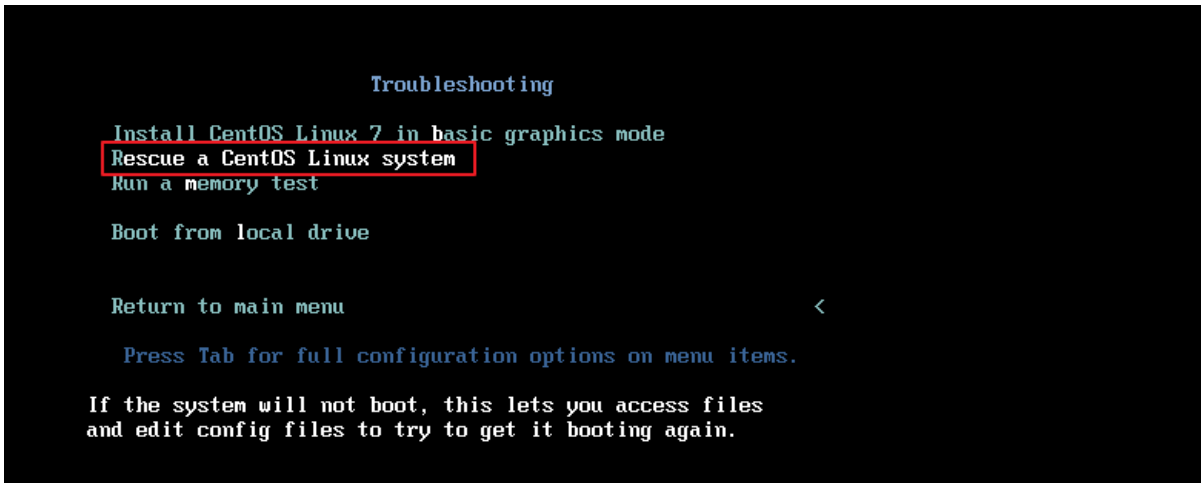
第二步：重启计算机

由于我的虚拟机挂载了光盘，所以一重启就进入光盘启动的界面



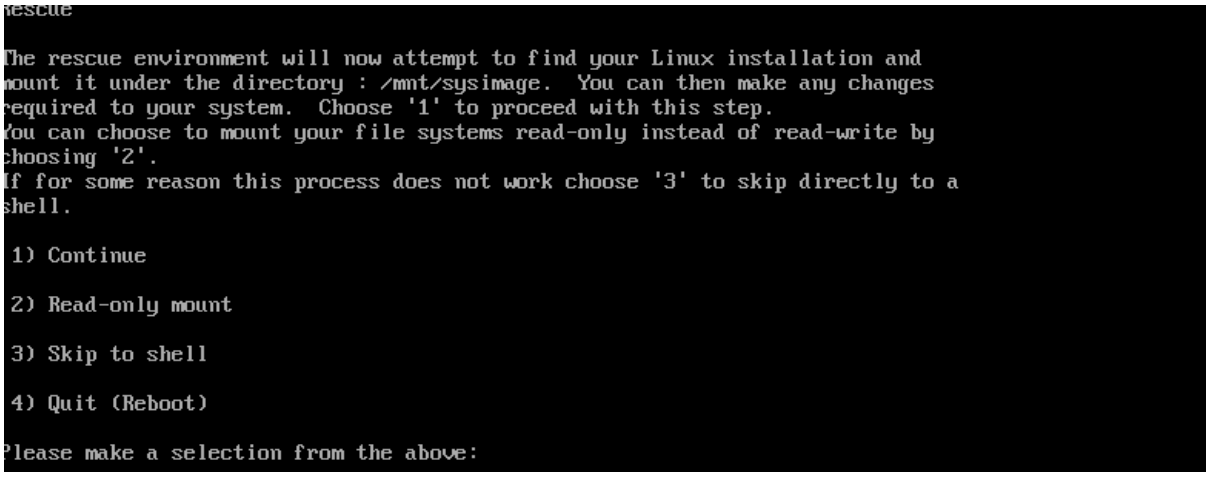
上面有三项，我们选择第三项进入troubleshooting

进入第三项后，点击第二项，进入救援模式的centos的系统



然后我们进入如下模式：

选择1，继续进行：



接下来，我们就会进入到一个shell模式中，不需要切根，进行系统修复：

```
If you would like to make your system the root environment, run the command:

        chroot /mnt/sysimage
Please press <return> to get a shell.
When finished, please exit from the shell and your system will reboot.
sh-4.2#
sh-4.2#
```

修复过程：

```
chroot  /mnt/sysimage/
grub2-install /dev/sda      # CentOS6的话就用grub-install  /dev/sda
```

```
sh-4.2#
sh-4.2# grub2-install --root-directory=/mnt/sysimage/ /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.
sh-4.2#
[anaconda] 1:main* 2:shell  3:log  4:storage-log  5:program-log      Switch tab: Alt+Tab | H
```

我们来查看一下：

hexdump -C -n 512 /dev/sda

```
sh-4.2# hexdump -C -n 512 /dev/sda
00000000  eb 63 90 00 00 00 00 00  00 00 00 00 00 00 00 00  |.c..............|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000050  00 00 00 00 00 00 00 00  00 00 00 80 01 00 00 00  |................|
00000060  00 00 00 00 ff fa 90 90  f6 c2 80 74 05 f6 c2 70  |...........t...p|
00000070  74 02 b2 80 ea 79 7c 00  00 31 c0 8e d8 8e d0 bc  |t....y|..1......|
00000080  00 20 fb a0 64 7c 3c ff  74 02 88 c2 52 be 05 7c  |. ..d|<.t...R..||
00000090  b4 41 bb aa 55 cd 13 5a  52 72 3d 81 fb 55 aa 75  |.A..U..ZRr=..U.u|
000000a0  37 83 e1 01 74 32 31 c0  89 44 04 40 88 44 ff 89  |7...t21..D.@.D..|
000000b0  44 02 c7 04 10 00 66 8b  1e 5c 7c 66 89 5c 08 66  |D.....f..\|f.\.f|
000000c0  8b 1e 60 7c 66 89 5c 0c  c7 44 06 00 70 b4 42 cd  |..`|f.\..D..p.B.|
000000d0  13 72 05 bb 00 70 eb 76  b4 08 cd 13 73 0d 5a 84  |.r...p.v....s.Z.|
000000e0  d2 0f 83 de 00 be 85 7d  e9 82 00 66 0f b6 c6 88  |.......}...f....|
000000f0  64 ff 40 66 89 44 04 0f  b6 d1 c1 e2 02 88 e8 88  |d.@f.D..........|
00000100  f4 40 89 44 08 0f b6 c2  c0 e8 02 66 89 04 66 a1  |.@.D.......f..f.|
00000110  60 7c 66 09 c0 75 4e 66  a1 5c 7c 66 31 d2 66 f7  |`|f..uNf.\|f1.f.|
00000120  34 88 d1 31 d2 66 f7 74  04 3b 44 08 7d 37 fe c1  |4..1.f.t.;D.}7..|
00000130  88 c5 30 c0 c1 e8 02 08  c1 88 d0 5a 88 c6 bb 00  |..0........Z....|
00000140  70 8e c3 31 db b8 01 02  cd 13 72 1e 8c c3 60 1e  |p..1......r...`.|
00000150  b9 00 01 8e db 31 f6 bf  00 80 8e c6 fc f3 a5 1f  |.....1..........|
00000160  61 ff 26 5a 7c be 80 7d  eb 03 be 8f 7d e8 34 00  |a.&Z|..}....}.4.|
00000170  be 94 7d e8 2e 00 cd 18  eb fe 47 52 55 42 20 00  |..}.......GRUB .|
00000180  47 65 6f 6d 00 48 61 72  64 20 44 69 73 6b 00 52  |Geom.Hard Disk.R|
00000190  65 61 64 00 20 45 72 72  6f 72 0d 0a 00 bb 01 00  |ead. Error......|
```

重启，修复完成。

实验：在centOS7下删除grub2下文件，并修复

第一步：删除grub2

```
rm -rf grub2
```

第二步，重启计算机

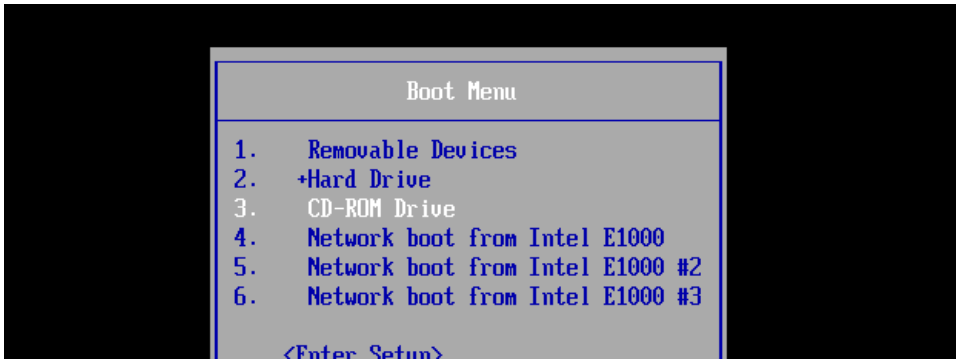
进入如下界面：

```
error: file '/grub2/i386-pc/normal.mod' not found.
Entering rescue mode...
grub rescue> _
```

接下来，我们重启系统，按Esc，进入光盘救援模式：

选择第三项，进入光盘救援（前提是挂载光盘）

以下步骤到进入shell同实验一相同，不在过多演示；



进入救援模式后：

第一步：切根

```
sh-4.2# chroot /mnt/sysimage/
bash-4.2# ls
1    bakup  boot  etc   lib    media  mnt  opt   root  sbin  sys     testdir  usr
app  bin    dev   home  lib64  misc   net  proc  run   srv   system  tmp      var
bash-4.2#
```

然后执行命令 （恢复grub）

grub2-install /dev/sda　　# CentOS6的话就用grub-install /dev/sda

```
bash-4.2# grub2-install /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.
bash-4.2#
```

下图中，我们可以看到在grub2文件夹中，还没有grub.cfg文件，接下来，我们需要生成：

```
bash-4.2# cd /boot
bash-4.2# ls grub2/
fonts  grubenv  i386-pc  locale
bash-4.2# _
```

缺少grub.cfg

生成配置文件：

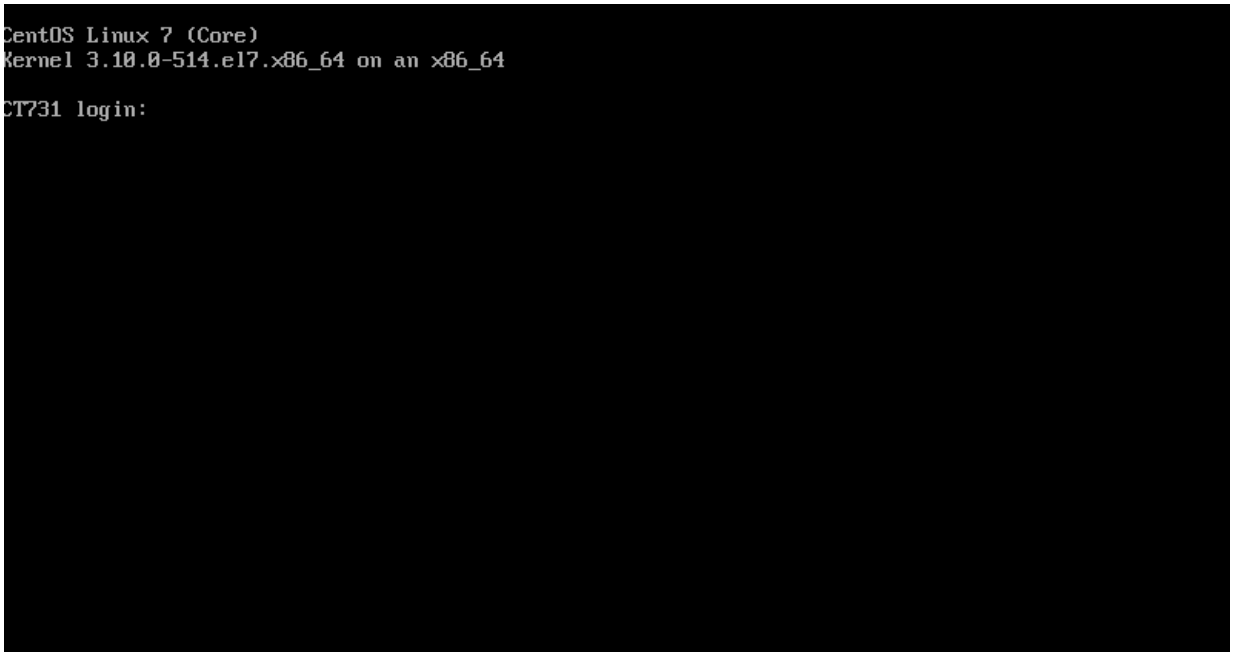
进入到grub2这个目录下，

grub2-mkconfig -o ./grub.cfg （CentOS7可以自动生成，CentOS6的vi /boot/grub/grub.conf 手动恢复了）

```
bash-4.2# grub2-mkconfig -o grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-514.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-514.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-618d3baeb5754f7092242f811c7aa740
Found initrd image: /boot/initramfs-0-rescue-618d3baeb5754f7092242f811c7aa740.img
done
bash-4.2# ls
fonts  grub.cfg  grubenv  i386-pc  locale
bash-4.2#
```

exit然后，

重启电脑：


CentOS Linux 7 (Core)
Kernel 3.10.0-514.el7.x86_64 on an x86_64
CT731 login:

完成！