操作系统实验二——用grub引导操作系统映像

> PB15000102 王嵩超

实验目的

使用Linux环境下的各种命令行工具、grub文件和Linux源码制作一个可引导硬盘映像，并用虚拟机引导。

实验过程

1. 编译Linux内核

在www.kernel.org下载内核源代码并解压。

> Linux内核有多个版本，为减少兼容问题，我选择了适中的3.10.105版本。

配置.configure

在解压后的根目录下打开终端，输入：

```
make i386_defconfig
```

make将会生成.configure文件。

```
songchaow@songchaow-pc:~/Linuxsrc/linux-3.10.105$ make i386_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

我们无需另生成根文件系统映像，故不用对.config文件做改动。

使用make编译

```
make -j 4
```

-j可使用四核并行编译，加快速度。

```
songchaow@songchaow-pc:~/Linuxsrc/linux-3.10.105$ make -j 4
make[1]: Nothing to be done for 'all'.
  CHK     include/generated/uapi/linux/version.h
make[1]: Nothing to be done for 'relocs'.
  CHK     include/generated/utsrelease.h
  CC      scripts/mod/devicetable-offsets.s
  GEN     scripts/mod/devicetable-offsets.h
  CALL    scripts/checksyscalls.sh
  HOSTCC  scripts/mod/file2alias.o
  HOSTLD  scripts/mod/modpost
  CHK     include/generated/compile.h
  CC      mm/page_alloc.o
  CC      fs/open.o
  BC      kernel/timeconst.h
  CC      arch/x86/kernel/quirks.o
  CC      kernel/ptrace.o
  CC      arch/x86/kernel/topology.o
  CC      arch/x86/kernel/kdebugfs.o
  CC      kernel/timer.o
  CC      fs/read_write.o
  CC      arch/x86/kernel/alternative.o
```

编译完成后会生成bzImage系统映像。Linux系统启动时由这个文件引导：

```
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Setup is 15000 bytes (padded to 15360 bytes).
System is 5351 kB
CRC d35c60f4
Kernel: arch/x86/boot/bzImage is ready  (#2)
songchaow@songchaow-pc:~/Linuxsrc/linux-3.10.105$
```

2. 制作根文件系统

> Linux内核引导时，会将一个"cpio"格式的压缩归档文件解压，并挂载到根目录，然后运行其中的init继续完成引导过程。本次实验init被替代成helloworld程序。

准备根目录文件

- 静态编译init.c

  init.c的内容如下：

```c
#include <stdio.h>
int main()
  {
    printf("helloworld!\n");
    return 0;
  }
```

使用gcc静态编译：

```
gcc -static -m32 init.c
```

> 64位Ubuntu系统在编译32位程序时会失败，原因是缺少gcc-multilib软件包。用apt-get命令安装后即可解决。

字符的输出需要用到字符设备和块设备。

```
songchaow@songchaow-pc:~/rootfs$ gcc -static -m32 init.c
songchaow@songchaow-pc:~/rootfs$ sudo mknod console c 5 1
[sudo] password for songchaow:
songchaow@songchaow-pc:~/rootfs$ sudo mknod ram b 1 0
songchaow@songchaow-pc:~/rootfs$ 
```

制作根目录映像

根目录映像须为cpio格式。

```
find .| cpio -o --format=newc > ../rootfs.img
```

现在可以用qemu检验是否能引导成功并输出helloworld：

```
QEMU
[    1.355733] Write protecting the kernel text: 6956k
[    1.355965] Write protecting the kernel read-only data: 2276k
helloworld!
[    1.391598] init (1) used greatest stack depth: 6440 bytes left
[    1.391735] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00
000000
[    1.391735]
[    1.391959] CPU: 0 PID: 1 Comm: init Not tainted 3.10.105 #2
[    1.392021] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS Ubunt
u-1.8.2-1ubuntu2 04/01/2014
[    1.392032]  c191a9c0 c191a9c0 c7851f2c c16bcf24 c7851f44 c16bc179 00000022 c
191a9c0
[    1.392032]  c784c000 c784c000 c7851f8c c1037562 c183d13c 00000000 c16c29f1 0
1852000
[    1.392032]  0849d5c8 000000d8 c7852000 c7ae2040 c7851fa4 c784c270 c784c270 c
7851f78
[    1.392032] Call Trace:
[    1.392032]  [<c16bcf24>] dump_stack+0x16/0x18
[    1.392032]  [<c16bc179>] panic+0x77/0x167
[    1.392032]  [<c1037562>] do_exit+0x9b2/0x9c0
[    1.392032]  [<c16c29f1>] ? apic_timer_interrupt+0x2d/0x34
[    1.392032]  [<c10383e0>] do_group_exit+0x30/0x90
[    1.392032]  [<c1038451>] SyS_exit_group+0x11/0x20
[    1.392032]  [<c16c8fba>] sysenter_do_call+0x12/0x12
```

可观察到出现`"helloworld!"`字样。

3. 制作grub启动软盘、启动硬盘

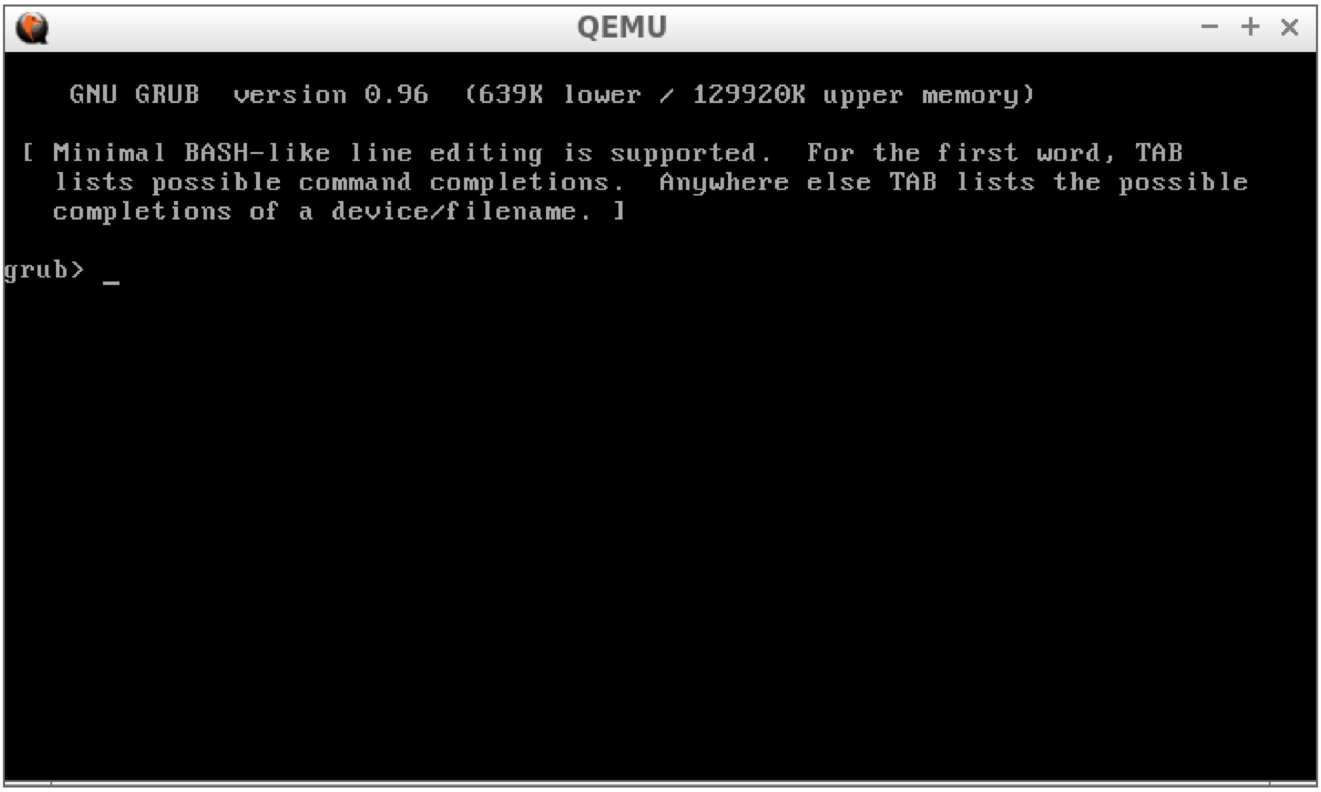
首先获取编译好了的i386平台的grub程序。

制作grub启动软盘

```
dd if=/dev/zero of=a.img bs=512 count=2880 #创建一个空映像文件
sudo losetup /dev/loop3 a.img #挂载为loop设备
sudo dd if=/home/songchaow/Downloads/grub-0.97-i386-pc/boot/grub/stage1 of=/dev/loop3 bs=5
count=1 #向映像文件内写入grub引导信息
sudo dd if=/home/songchaow/Downloads/grub-0.97-i386-pc/boot/grub/stage2 of=/dev/loop3 bs=5
seek=1
sudo losetup -d /dev/loop3 #删除loop设备
```

测试grub软盘是否可以启动：

```
qemu-system-i386 -fda a.img
```



制作grub启动硬盘

```
dd if=/dev/zero of=32M.img bs=4096 count=8192
sudo losetup /dev/loop3 32M.img
echo 在建立活动分区...
sudo fdisk /dev/loop3
sudo losetup -d /dev/loop3

sudo losetup -o 1048576 /dev/loop3 32M.img #重新将磁盘从分区开始处挂载 磁盘开始部分用于被grub写入引导信息
#1048576由扇区数2048与字节数512相乘而得
sudo mke2fs /dev/loop3 #转换分区格式至ext2
sudo mount /dev/loop3 rootfs
```

fdisk的操作

- 输入n建立新分区
- 输入p，建立主分区
- 输入默认的2048作为起始扇区，65535作为末扇区。
- 输入w将改动写入32M.img

32M.img挂载到rootfs后，将bzImage和myinitrd4M.img拷贝到rootfs中。

将grub-i386/boot/grub目录下所有文件拷贝到rootfs/boot/grub目录下：

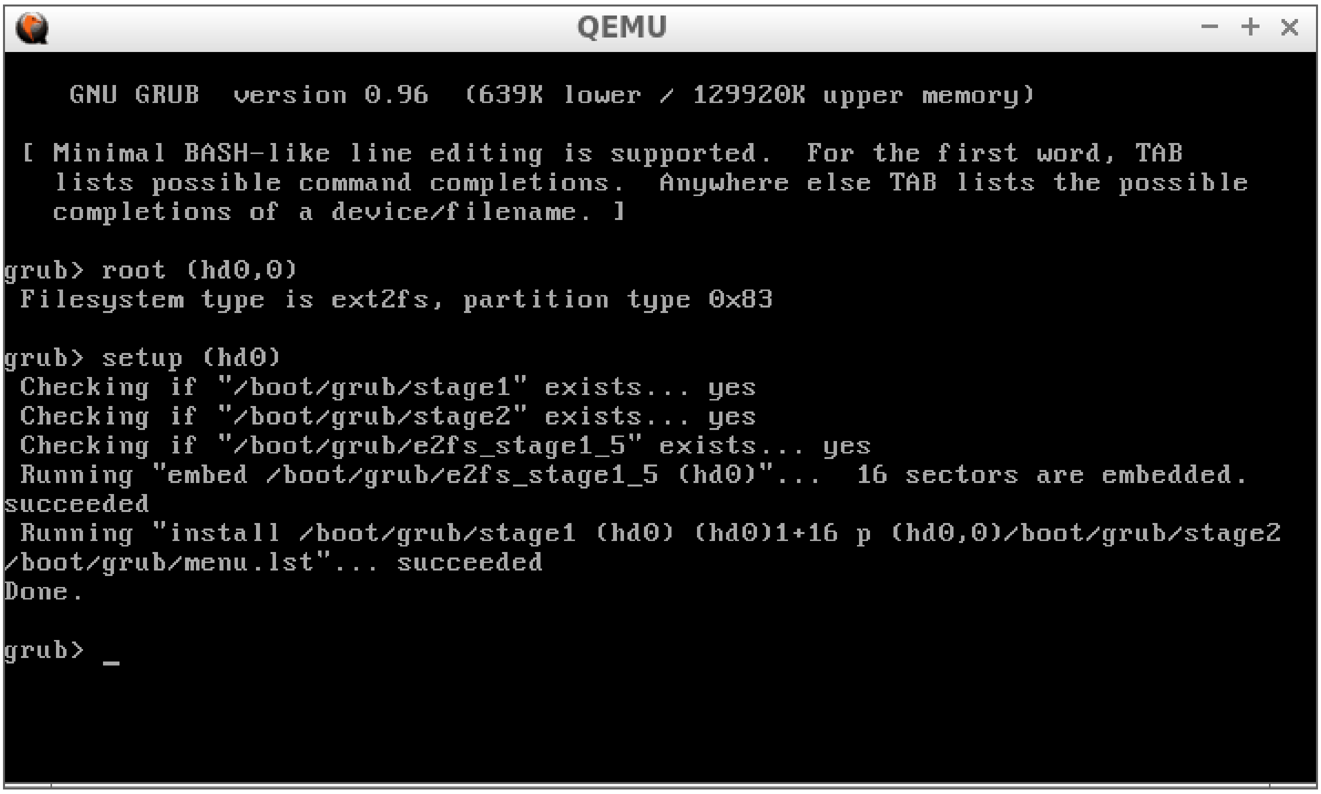
```
sudo mkdir rootfs/boot
sudo mkdir rootfs/boot/grub
sudo cp /home/songchaow/Downloads/grub-0.97-i386-pc/boot/grub/* rootfs/boot/grub
```

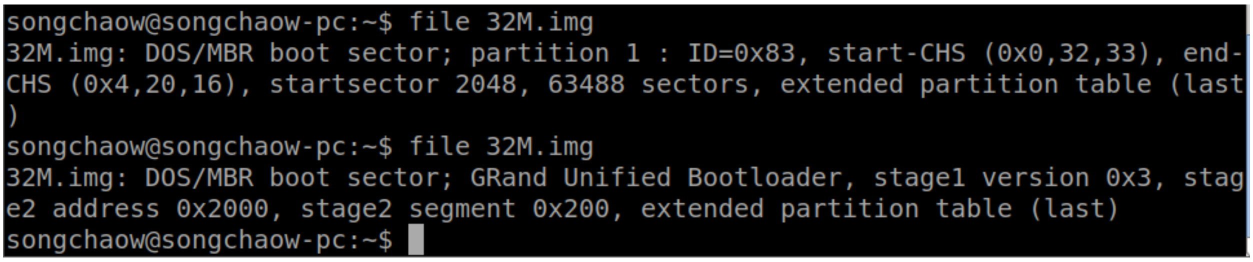
编写启动菜单menu.lst：

```
default 0
timeout 30
title linux on 32M.img
root (hd0,0)
kernel (hd0,0)/bzImage root=/dev/ram init=/bin/ash
initrd (hd0,0)/rootfs.img
```

利用grub启动软盘，在硬盘映像上添加grub功能

```
qemu-system-i386 -boot a -fda a.img -hda 32M.img
```

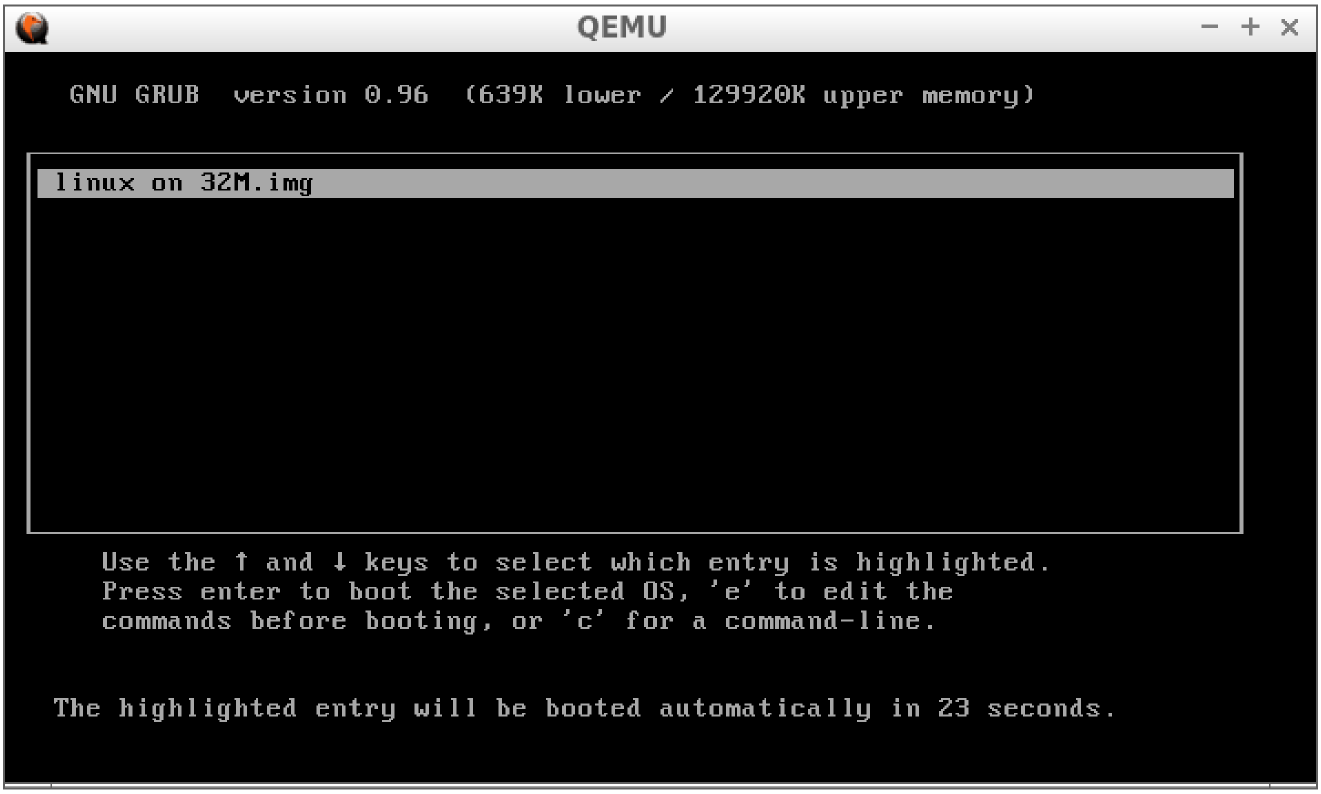


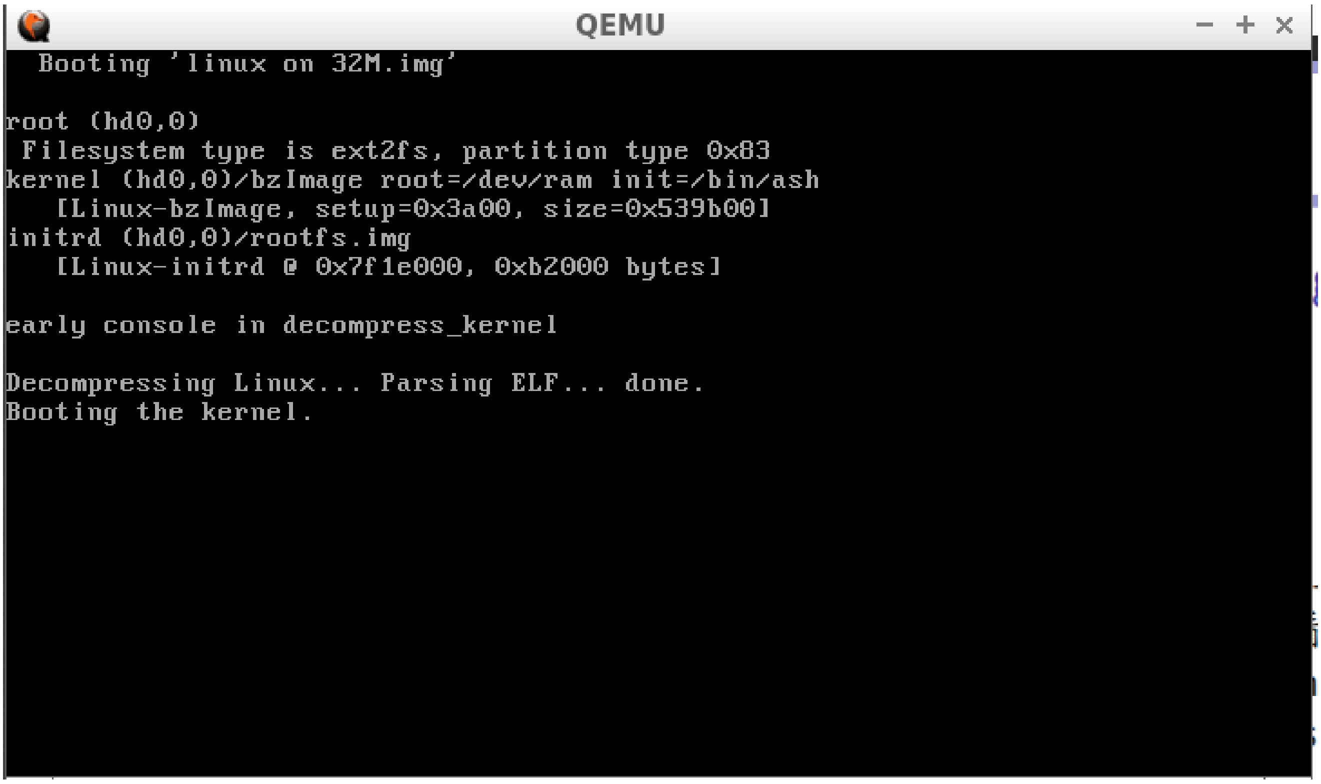
> 顺便，我在硬盘映像上添加grub功能的前后用file命令查看了32M.img镜像文件的改变。
>
> 上行为安装前，下行为安装后。可发现引导信息已经改变。
>
> 
>

最后，测试从硬盘grub启动：

```
qemu-system-i386 -hda 32M.img
```

过程展示：


QEMU
GNU GRUB version 0.96 (639K lower / 129920K upper memory)
linux on 32M.img
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.
The highlighted entry will be booted automatically in 23 seconds.


QEMU
Booting 'linux on 32M.img'
root (hd0,0)
Filesystem type is ext2fs, partition type 0x83
kernel (hd0,0)/bzImage root=/dev/ram init=/bin/ash
[Linux-bzImage, setup=0x3a00, size=0x539b00]
initrd (hd0,0)/rootfs.img
[Linux-initrd @ 0x7f1e000, 0xb2000 bytes]
early console in decompress_kernel
Decompressing Linux... Parsing ELF... done.
Booting the kernel.