

OS:Lab02-制作 grub 启动软盘

Stu:金泽文 No:PB15111604

实验目的：

通过制作 grub 启动软盘，理解 linux 启动引导的机制。

实验内容：

1. 准备模拟器 qemu
2. 制作带 grub 的磁盘映像
3. 利用 qemu 和 gdb 调试 linux

具体过程及相关原理、命令解释：

一、 准备模拟器 qemu

·安装 qemu

通过 `sudo apt-get install qemu` 安装 qemu

通过 `sudo ln -s /usr/bin/qemu-system-i386`

`/usr/bin/qemu` 得到 qemu 链接以方便访问

·下载 linux 并编译

在 google 中得到 3.10.98 的下载链接，

由于我下载的格式为 xz，`tar -xvf` 出错，google 之后使用

`xz` 正确解压缩

·制作根目录系统

按照教程，用 `mknod` 命令在 `./dev/` 目录下为设备 `console` 和

ram 分配主设备号与此设备号。(由于 linux 将设备抽象为文件 , 故设备号通过修改文件符号标志设备的类型)。

写一个 init 程序并用-static -m32 编译。

期间 , 遇到插曲 :

```
→ Documents gcc -o init first.c -static -m32
In file included from /usr/include/stdio.h:27:0,
               from first.c:1:
/usr/include/features.h:367:25: fatal error: sys/cdefs.h: 没有那个文件或目录
compilation terminated.
→ Documents gcc -o init first.c -static -m32
In file included from /usr/include/stdio.h:27:0,
```

经过 google , 发现少 32 位 lib , 通过 `sudo apt-get install libc6-dev-i386` 解决。

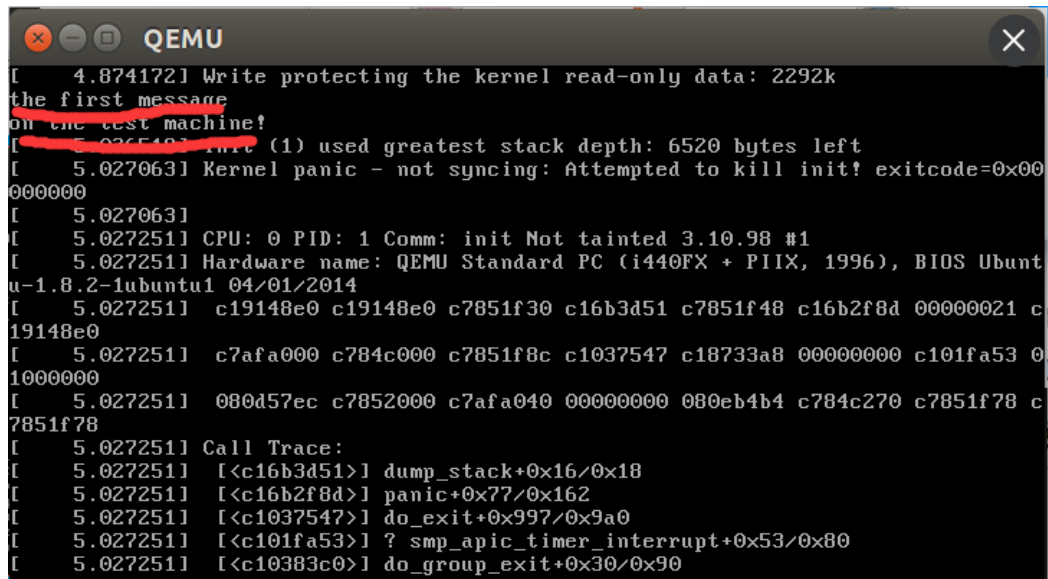
通过 cpio 将文件信息传入到 rootfs.img 镜像中。所用的格式 newc 是新兴 (SVR4) 跨平台格式 , 支持大于 65536i 节点的文件系统 , 正好适用于我们的 rootfs。

·运行内核

由于前面助教提醒 , 这一步基本没有走弯路。

只不过一开始没有理解 PATH_to_bzImage 是什么意思并且将大写的 i 看成了小写的 L , 浪费了十多分钟时间。

这一步得到：



```
[ 4.8741721] Write protecting the kernel read-only data: 2292k
the first message
on the test machine!
[ 5.0270631] init (1) used greatest stack depth: 6520 bytes left
[ 5.0270631] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[ 5.0270631]
[ 5.0272511] CPU: 0 PID: 1 Comm: init Not tainted 3.10.98 #1
[ 5.0272511] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS Ubuntu-1.8.2-1ubuntu1 04/01/2014
[ 5.0272511] c19148e0 c19148e0 c7851f30 c16b3d51 c7851f48 c16b2f8d 00000021 c19148e0
[ 5.0272511] c7afa000 c784c000 c7851f8c c1037547 c18733a8 00000000 c101fa53 01000000
[ 5.0272511] 080d57ec c7852000 c7afa040 00000000 080eb4b4 c784c270 c7851f78 c7851f78
[ 5.0272511] Call Trace:
[ 5.0272511] [] dump_stack+0x16/0x18
[ 5.0272511] [] panic+0x77/0x162
[ 5.0272511] [] do_exit+0x997/0x9a0
[ 5.0272511] [] ? smp_apic_timer_interrupt+0x53/0x80
[ 5.0272511] [] do_group_exit+0x30/0x90
```

这里解释一下 `qemu -kernel bzImage -initrd rootfs.img -append " root=/dev/ram rdinit=init"` 各个指令的含义。

Qemu 当然是运行内核的模拟器。

`bzImage` 是压缩的内核映像，`bz` 表示 “big zImage”，

`initrd` 则表示 Linux 的初始 Ram 磁盘，是在系统引导的过程中挂载的临时根文件系统，所有后面跟的是 `rootfs.img`，

跟着是 `root = /dev/ram`，表示 `root` 设备是 `/dev/ram`，

`rdinit` 用来指定 `ramdisk_execute_command`，以通知系统执行 `init_post()`，在我们的例子中，就会首先执行可执行文件 `init`（类似于真是 linux 系统中的 `init` 父进程）。

二、制作带 grub 的磁盘映像

·获取 grub。

Grub--多系统引导管理器。

·建立软盘映像

```
dd if=/dev/zero of=a.img bs=512 count=2880
```

由 `man dd` 得到 dd 的相关信息：convert and copy a file.

If 表示 read from file 而不是 stdin ,

of 表示 write to file 而不是 stdout。

bs 表示一次读的字节数，为 512 字节

count 表示读的次数。

而/dev/zero 负责提供 0

可见这个命令只是为了开辟 a.img 这样大小的空间

·添加 grub 启动功能

```
sudo losetup /dev/loop3 a.img
```

```
sudo dd if=./grub-0.97-i386-pc/boot/grub/stage1
```

```
of=/dev/loop3 bs=512 count=1
```

```
sudo dd if=./grub-0.97-i386-pc/boot/grub/stage2
```

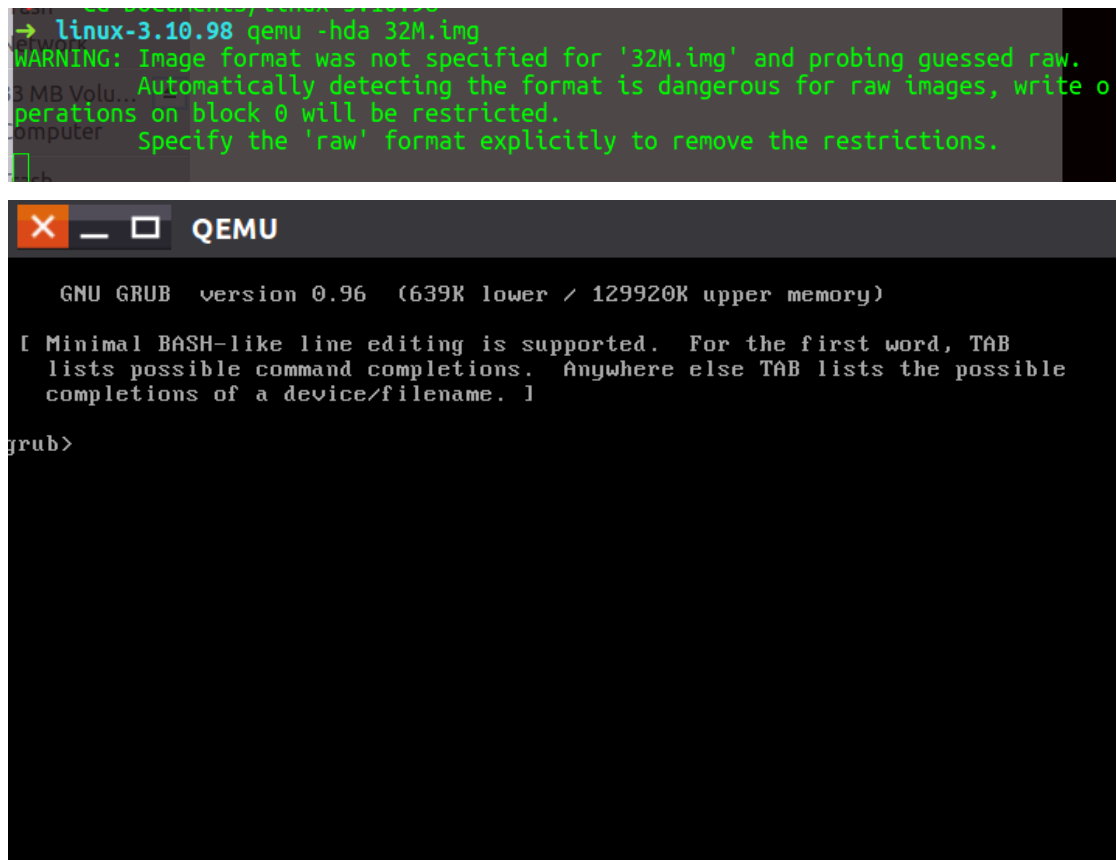
```
of=/dev/loop3 bs=512 seek=1
```

```
sudo losetup -d /dev/loop
```

在类 unix 系统里，loop 设备是一种伪设备，能使我们像块设备一样访问文件。

一个 loop 设备必须和一个文件进行连接，比如这里，
/dev/loop3 与 a.img 连接，这个 a.img 映像文件就像一个磁盘设备一样被挂载（叫 loop mount），这里借助 dd 和 losetup 的方式，将./grub-0.97-i386-pc/boot/grub/中对应文件信息按块输出到我们的 a.img 上。

运行 `qemu -hda 32M.img` 后 ,



```
linux-3.10.98 qemu -hda 32M.img
WARNING: Image format was not specified for '32M.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

GNU GRUB  version 0.96  (639K lower / 129920K upper memory)

[ Minimal BASH-like line editing is supported.  For the first word, TAB
  lists possible command completions.  Anywhere else TAB lists the possible
  completions of a device/filename. ]

grub>
```

正常。

解释：`qemu -hda 32M.img` 是将 32M.img 作为软盘 0 的
镜像

·准备磁盘映像

```
dd if=/dev/zero of=32M.img bs=4096 count=8192
```

```
sudo losetup /dev/loop3 32M.img
```

在磁盘映像上建立一个活动分区

```
sudo fdisk /dev/loop3 ,
```

`fdisk` 作为磁盘管理工具，可以用 `m` 查看 `help`，经过 `n`，一
系列默认之后就建立了一个 `partition` 分区，这一步其实是在

32M.img 分为一个区，并将其设置为引导扇区 MBR。

```
→ grub-0.97-i386-pc sudo fdisk /dev/loop3
Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x7c30e1fc.

命令(输入 m 获取帮助): █
```

`sudo losetup -d /dev/loop3` 卸载映像

·将活动分区格式化成 ext2fs，并 mount 到 rootfs 目录上

`sudo losetup -o $((2048*512)) /dev/loop3 32M.img`

其中，`$((2048*512))`是分区的起始位置,通过-o 选项设置数据偏移量，因为前面`$((2048*512))`是 MBR 内容。其中，2048 是通过 file 32M.img 得到的 startsector 信息。

`sudo mke2fs /dev/loop3`

这里 mke2fs 即为格式化命令。

`sudo mount /dev/loop3 rootfs`

挂载活动分区到 rootfs 上。

将前面制作的 bzImage 和 rootfs.img 拷贝到 rootfs 中。

·添加 grub 功能

准备目录，拷贝文件

`sudo mkdir rootfs/boot`

`sudo mkdir rootfs/boot/grub`

```
sudo cp ./grub-0.97-i386-pc/boot/grub/*rootfs/boot/grub
```

编写 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
```

default 0 代表 grub 的默认启动项为 0 号系统

timeout 30 若用户在 30s 中没有操作，则启动默认项

title 指定启动的操作系统菜单的名称

root 指定启动分区，此处为 (hd0,0) ,表示第一块硬盘的
一个分区。

Kernel 指定启动的内核的绝对路径和名称，**root**，**init** 为参
数

Initrd 指定根系统目录

·利用 grub 启动软盘，在硬盘映像上添加 grub 功能

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

```
GNU GRUB version 0.96 (639K lower / 129920K upper memory)

[ Minimal BASH-like line editing is supported. For the first word, TAB
  lists possible command completions. Anywhere else TAB lists the possible
  completions of a device/filename. ]

grub> root (hd0,0)
  Filesystem type is ext2fs, partition type 0x83

grub> setup (hd0)
  Checking if "/boot/grub/stage1" exists... yes
  Checking if "/boot/grub/stage2" exists... yes
  Checking if "/boot/grub/e2fs_stage1_5" exists... yes
  Running "embed /boot/grub/e2fs_stage1_5 (hd0)"... 16 sectors are embedded.
  succeeded
  Running "install /boot/grub/stage1 (hd0) (hd0)1+16 p (hd0,0)/boot/grub/stage2
  /boot/grub/menu.lst"... succeeded
  Done.

grub>
```

Root(hd0,0) 将根分区设置为磁盘第一分区

Setup(hd0) 自动安装 grub

·测试从磁盘启动进入 grub 界面

```
→ linux-3.10.98 bash
zevin@ubuntu:~/Documents/linux-3.10.98$ qemu -hda 32M.img
WARNING: Image format was not specified for '32M.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

QEMU - Press Ctrl-Alt to exit mouse grab

GNU GRUB version 0.96 (639K lower / 129920K upper memory)

linux on 32M.img
```

成功！


```
→ linux-3.10.98 bash
zevin@ubuntu:~/Documents/linux-3.10.98$ qemu -hda 32M.img
WARNING: Image format was not specified for '32M.img' and probing
Automatically detecting the format is dangerous for raw i
operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restric
main-loop: WARNING: I/O thread spun for 1000 iterations
[
[ 17.267614] Write protecting the kernel read-only data: 2292k
[ 17.419095] hrtimer: interrupt took 4351137 ns
the first message
on the test machine!
[ 18.286115] init (1) used greatest stack depth: 6452 bytes left
[ 18.286115] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00
000000
[ 18.286115]
[ 18.286115] CPU: 0 PID: 1 Comm: init Not tainted 3.10.98 #1
[ 18.286115] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS Ubunt
u-1.8.2-1ubuntu1 04/01/2014
[ 18.286115] c19148e0 c19148e0 c7851f30 c16b3d51 c7851f48 c16b2f8d 00000022 c
18148e0
```

三、 利用 qemu 和 gdb 调试 linux

`qemu -kernel arch/x86/boot/bzImage -s -S`

-s 设置 remote 端口 1234

-S 设置开启时中断

·打开 gdb

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote: 1234
1234: 没有那个文件或目录.
(gdb) target remote: 1234
: 1234: 没有那个文件或目录.
(gdb) target remote1234
Undefined target command: "remote1234". Try "help target".
(gdb) target remote:1234
Remote debugging using :1234
0x0000ffff in ?? ()
(gdb) c
Continuing.
```

```
(gdb) file vmlinux
Reading symbols from vmlinux...(no debugging symbols found)...done.
(gdb) target remote:1234
Remote debugging using :1234
0x00000fff0 in ?? ()
(gdb) break start_kernel
Breakpoint 1 at 0xc196c650
(gdb) c
Continuing.
Breakpoint 1, 0xc196c650 in start_kernel ()
(gdb) []
```

QEMU [Stopped]

SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92300+07ED2300 C980

Booting from ROM...

early console in decompress_kernel

Decompressing Linux... Parsing ELF... done.

Booting the kernel.

没有携带调试信息，故重新配置 linux，

```
.config - Linux/x86 3.10.98 Kernel Configuration
> Kernel hacking

Kernel hacking
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

[*] Lock usage statistics
[ ] Sleep inside atomic section checking
[ ] Locking API boot-time self-tests
[*] Stack utilization instrumentation
[ ] kobject debugging
[ ] Highmem debugging
[*] Compile the kernel with debug info
[ ] Reduce debugging information
[ ] Debug VM
[ ] Debug VM translations

<Select> < Exit > < Help > < Save > < Load >
```

再重新编译。

实验吐槽与总结：

首先贴上我在群里对本次实验的吐槽。

另外，我也很想吐槽一下这个实验。@xlanchen 。好几个英才班的同学羡慕我们学了很多东西，做了很多东西。但是我个人感觉，我们敲了很多东西，并且解决了一些遇到的小问题，但是实际上很多人没有学到很多东西，（这里很钦佩送小牛同学）。我记得老师当时说过（可能不那么认真的）一句话：这个实验你就是照着敲一遍也能学到东西。首先，这里我不是很清楚我们的实验二，老师到底要求或者期望我们达到怎样的一个水平，对 grub 引导机制这个原理的理解程度。其次，整个教程都是教我们敲命令。我比较早地完成了实验，对原理稀里糊涂，好多同学问我，他们也表示稀里糊涂。为什么呢？我想是因为没有强调原理啊。哪怕老师之前讲了一些启动的东西，但是挂载是什么，生成映像怎么实现，都没有说。这些如果本来就要求我们自己找资料自己学，那也理解，但是教程本身没有反映这样一种需求。再其次，我想知道能不能不直接给我们“所有的”命令，一是直接敲上去没理解没意思，二是很多命令教程里或刻板或易误解。倒不如让我们自己想自己推更有帮助。以上是我的看法，当然可能都是错的，不过都跟老师反映一下。（我去吃早饭了）

总结与反思：

在这个实验中，我学会了很多解决问题的方法。csdn，man，google，stackoverflow，与同学分享遇到的错误与解决方法等等；同时，我学会并熟练掌握了很多 shell 语法，习惯了命令行

的环境。对于实验本身，我加深了对 grub 引导系统启动整个机制的理解，不过让我逐字逐句地、按部就班地、仔仔细细地说出整个机制的细节，还有许多不足。

实验中遇到的各种脑残问题，除了难以启齿的智障问题，都在前面的实验步骤中有所指明。

最后，整个实验以及吐槽后的讨论，我更加明白了一个道理：

computer science 或者 information technology 行业的人，确实需要强硬的搜集信息，解决问题的能力。今后应该更加锻炼这种能力！