

Linux 内核编译及运行

一、实验目的

- 1、选择合适平台工具，搭建环境，能够对 Linux0.11 内核进行编译，并在模拟器中运行。
- 2、能够通过自主学习，解决在内核编译运行中的问题。

二、主要平台和工具

本操作系统实验的硬件环境是 IA-32 (x86) 架构的 PC 机，主要软件环境是 Bochs + gcc + 你最喜欢的编辑器 / IDE + 你最喜欢的操作系统 + Linux 0.11 源代码。

实验的基本流程是根据实验要求编写应用程序、修改 Linux 0.11 的源代码，用 gcc 编译后，在 Bochs 的虚拟环境中运行、调试目标代码。

上述实验环境涉及到的软件都是免费且开源的，具有较强的可移植性，可以在多种计算机的多种操作系统上搭建。为方便实验者，在最常见的平台 Ubuntu(最流行的 GNU/Linux 发行版之一)——上制作了 hit-oslab 集成环境，它基本包含了实验所需的所有软件，直接在 LINUX 平台解压使用。

(1) X86 模拟器 Bochs

Bochs 是一个免费且开放源代码的 IA-32 (x86) 架构 PC 机模拟器。在它模拟出的环境中可以运行 Linux、DOS 和各种版本的 Windows 等多种操作系统。而 Bochs 本身具有很高的移植性，可以运行在多种软硬件平台之上，这也是我们选择它做为指定模拟器的主要原因。

如果您想拥抱自由的 Linux，那么 Bochs 几乎是您的不二选择。如果您想继续把自己绑定在 Windows 平台上，那么除了 Bochs，您还可以选用 VMware 或者 Microsoft Virtual PC。它们是最著名虚拟机软件，而且都可以免费使用。因为 Bochs 的是模拟器，其原理决定了它的运行效率会低于虚拟机。Bochs 有虚拟机无可比拟的调试操作系统的能力，所以我们更建议您选用 Bochs。hit-oslab 已经内置了 bochs，本实验后文假定的缺省环境也是 Bochs。

关于 Bochs 的更详细的介绍请访问它的 [主页](#) 及 Bochs 使用手册。

(2) GCC 编译器

GCC 是和 Linux 一起成长起来的编译器。Linux 最初的版本就是由 GCC 编译的。现在 GCC 也是在自由软件领域应用最广泛的编译器。所以，我们也选择 GCC 做为本书实验的指定编译器。

(3) GDB 调试器

GDB 调试器是 GCC 编译器的兄弟。做为自由软件领域几乎是唯一的调试器，它秉承了 Unix 类操作系统的一贯风格，采用纯命令行操作，有点儿类似 dos 下的 debug。

关于它的使用方法请看 GDB 使用手册。

另外，可以学习实验楼的《GDB 简明教程》，通过动手实验学习 Linux 上 GDB 调试 C 语言程序的基本技巧。

(4) Ubuntu (GNU/Linux)

Ubuntu 也许不是目前最好用的 Linux 桌面发行版，但它一定是最流行的。主要特点是易用，非常的易用。现在，已经有越来越多的人开始用 Ubuntu 完全代替 Windows，享受更加自由、安全、守法的感觉。

Ubuntu 的主页是 <http://www.ubuntu.com/>，这里可以免费下载到 iso 文件。

我们强烈建议您在 Ubuntu 下做实验。因为有些实验内容涉及到在自己改进的 Linux 0.11 下，运行自己编的应用程序。被改进的功能都是高版本 Linux 内核已经具有的，在其上确认自己编写的应用程序无误后，再用之测试自己改进的 Linux 0.11，可以更有信心些。

三、实验任务

- 1、在蓝桥云上完成实验 1 熟悉实验环境，完成内核的编译运行、调试、文件交换等。
- 2、在自己电脑上安装虚拟机、Linux、搭建实验环境，完成内核的编译运行，熟悉调试和文件交换。

四、实验环境的工作模式

(1) 准备环境

hit-oslab 实验环境简称 oslab，是一个压缩文件 (hit-oslab-linux-20110823.tar.gz)，这个文件请 COPY 或下载到你的用户主目录下的 oslab 目录（假如你的用户主目录 /home/shiyanlou）下，大家可以使用下面的命令解压展开压缩包即可工作。

推荐大家使用如下的命令解压到 /home/shiyanlou/oslab/ 中。

```
# 进入到 oslab 所在的文件夹
$ cd /home/shiyanlou/oslab/

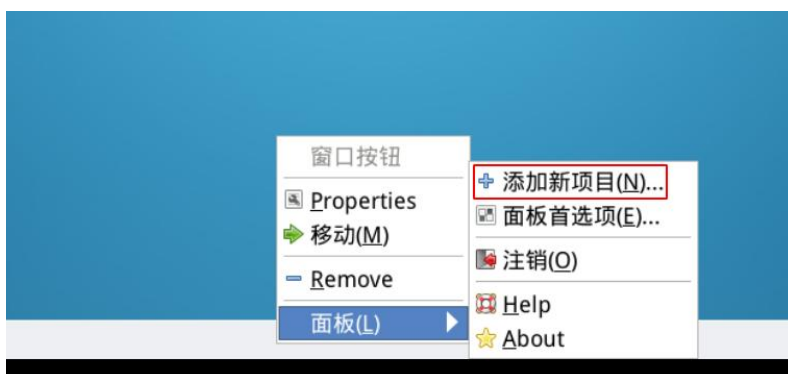
# 解压，并指定解压到 /home/shiyanlou/
# 这样的话，在 /home/shiyanlou/oslab/ 中就能找到解压后的所有文件
$ tar -zxvf hit-oslab-linux-20110823.tar.gz \
  -C /home/shiyanlou/

# 查看是否解压成功
$ ls -al
# 除了压缩包 hit-oslab-linux-20110823.tar.gz 之外，其他的就是压缩包
  中的内容
```

注：命令行最后的“\”表示该行命令没结束，如果命令在一行中输入完毕，则不需要命令为：tar -zxvf hit-oslab-linux-20110823.tar.gz

在自己电脑搭建环境执行命令时注意报错情况，要有 root 权限，可增加该用户 root 权限或切换到 root 用户再执行该命令

如果终端窗口最小化后无法找到，可以在任务栏右键，面板 -> 添加新项目 -> 窗口菜单 -> Add 来打开显示。



(2) 文件结构

➤ Image 文件

oslab 工作在一个宿主操作系统之上，我们使用的 Linux，在宿主操作系统之上完成对 Linux 0.11 的开发、修改和编译之后，在 linux-0.11 目录下会生产一个名为 Image 的文件，它就是编译之后的目标文件。

该文件内已经包含引导和所有内核的二进制代码。如果拿来一张软盘，从它的 0 扇区开始，逐字节写入 Image 文件的内容，就可以用这张软盘启动一台真正的计算机，并进入 Linux 0.11 内核。

oslab 采用 bochs 模拟器加载这个 Image 文件，模拟执行 Linux 0.11，这样省却了重新启动计算机的麻烦。

➤ bochs 目录

bochs 目录下是与 bochs 相关的执行文件、数据文件和配置文件。

➤ run 脚本

run 是运行 bochs 的脚本命令。

运行后 bochs 会自动在它的虚拟软驱 A 和虚拟硬盘上各挂载一个镜像文件，软驱上挂载是 linux-0.11/Image，硬盘上挂载的是 hdc-0.11.img。

因为 bochs 配置文件中的设置是从软驱 A 启动，所以 Linux 0.11 会被自动加载。

而 Linux 0.11 会驱动硬盘，并 mount 硬盘上的文件系统，也就是将 hdc-0.11.img 内镜像的文件系统挂载到 0.11 系统内的根目录 —— /。在 0.11 下访问文件系统，访问的就是 hdc-0.11.img 文件内虚拟的文件系统。

➤ hdc-0.11.img 文件

hdc-0.11.img 文件的格式是 Minix 文件系统的镜像。

Linux 所有版本都支持这种格式的文件系统，所以可以直接在宿主 Linux 上通过 mount 命令访问此文件内的文件，达到宿主系统和 bochs 内运行的 Linux 0.11 之间交换文件的效果。

hdc-0.11.img 内包含有：

- Bash shell;
- 一些基本的 Linux 命令、工具，比如 cp、rm、mv、tar;
- vi 编辑器;
- gcc 1.4 编译器，可用来编译标准 C 程序;
- as86 和 ld86;
- Linux 0.11 的源代码，可在 0.11 下编译，然后覆盖现有的二进制内核。

其他文件在后面用到的时候会进行单独讲解。

四、使用方法

开始使用之前的准备活动：把当前目录切换到 oslab 下，用 pwd 命令确认，用 ls -l 列目录内容。

```
# 切换目录
$ cd /home/shiyanlou/oslab/

# 确认路径
$ pwd

# 查看目录内容
$ ls -l
```

本实验的所有内容都在本目录或其下级目录内完成。

```

shiyanolou@82a7ecabb02f:~$ cd /home/shiyanolou/oslab/
shiyanolou@82a7ecabb02f:~/oslab$ pwd
/home/shiyanolou/oslab
shiyanolou@82a7ecabb02f:~/oslab$ ls -l
总用量 95272
drwxr-xr-x  2 shiyanolou shiyanolou    4096  9月  5  2010 bochs
-rwxr-xr-x  1 shiyanolou shiyanolou    115  8月 30  2008 dbg-asm
-rwxr-xr-x  1 shiyanolou shiyanolou    119  8月 30  2008 dbg-c
-rwxr-xr-x  1 shiyanolou shiyanolou 12423461  8月 28  2008 gdb
-rw-r--r--  1 shiyanolou shiyanolou     75  8月 28  2008 gdb-cmd.txt
drwxr-xr-x  2 shiyanolou shiyanolou    4096  9月  5  2010 hdc
-rw-r--r--  1 shiyanolou shiyanolou 63504384 10月  2  2009 hdc-0.11.img
-rw-rw-r--  1 shiyanolou shiyanolou 21586449  1月 13  2015 hit-oslab-linux
-20110823.tar.gz
drwxr-xr-x 10 shiyanolou shiyanolou    4096  8月 23  2011 linux-0.11
-rwxr-xr-x  1 shiyanolou shiyanolou    126 10月  9  2008 mount-hdc
-rwxr-xr-x  1 shiyanolou shiyanolou    254  9月  1  2009 run
-rwxr-xr-x  1 shiyanolou shiyanolou    268  9月  1  2009 rungdb
shiyanolou@82a7ecabb02f:~/oslab$

```

(1) 编译内核

“编译内核”比“编写内核”要简单得多。

首先要进入 `linux-0.11` 目录，然后执行 `make` 命令：

```

$ cd ./linux-0.11/
$ make all

```

因为 `all` 是最常用的参数，所以可以省略，只用 `make`，效果一样。

在多个处理器的系统上，可以用 `-j` 参数进行并行编译，加快速度。例如双 CPU 的系统可以：

```

$ make -j 2

```

`make` 命令会显示很多很多的信息，你可以尽量去看懂，也可以装作没看见。只要最后几行中没有“error”就说明编译成功。

最后生成的目标文件是一个软盘镜像文件——`linux-0.11/Image`（下面的图中给出了详细的信息）。如果将此镜像文件写到一张 1.44MB 的软盘上，就可以启动一台真正的计算机。

```

shiyanolou@82a7ecabb02f:~/oslab/linux-0.11$ ls -al
总用量 292
drwxr-xr-x 10 shiyanolou shiyanolou    4096  6月 18 09:57 .
drwxr-xr-x  5 shiyanolou shiyanolou    4096  9月  5  2010 ..
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:56 boot
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:56 fs
-rw-rw-r--  1 shiyanolou shiyanolou 124068  6月 18 09:57 Image
drwxr-xr-x  5 shiyanolou shiyanolou    4096  9月  5  2010 include
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:56 init
drwxr-xr-x  5 shiyanolou shiyanolou    4096  6月 18 09:56 kernel
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:56 lib
-rw-r--r--  1 shiyanolou shiyanolou    3442  8月 23  2011 Makefile
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:56 mm
-rw-rw-r--  1 shiyanolou shiyanolou 10729  6月 18 09:56 System.map
-rw-r--r--  1 shiyanolou shiyanolou 110724 10月  9  2008 tags
drwxr-xr-x  2 shiyanolou shiyanolou    4096  6月 18 09:57 tools
shiyanolou@82a7ecabb02f:~/oslab/linux-0.11$

```


linux-0.11 目录下是全部的源代码，很多实验内容都是要靠修改这些代码来完成。修改后需要重新编译内核，还是执行命令：`make all`。

`make` 命令会自动跳过未被修改的文件，链接时直接使用上次编译生成的目标文件，从而节约编译时间。但如果重新编译后，你的修改貌似没有生效，可以试试先 `make clean`，再 `make all`（或者一行命令：`make clean && make all`。`make clean` 是删除上一次编译生成的所有中间文件和目标文件，确保是在全新的状态下编译整个工程）。

在编译内核时若出现如下错误：

```
root@yhy:/home/yhy/oslab/linux-0.11# make
as86 -0 -a -o boot/bootsect.o boot/bootsect.s
make: as86: Command not found
make: *** [Makefile:101: boot/bootsect] Error 127
```

则可查看 `as86` 情况：

```
root@yhy:/home/yhy/oslab/linux-0.11# apt-cache search as86
bin86 - 16-bit x86 assembler and loader
```

显示 `as86` 在 `bin86` 包，则安装该包：

```
root@yhy:/home/yhy/oslab/linux-0.11# sudo apt install bin86
```

若缺 `gcc-3.4`：

```
make: gcc-3.4: Command not found
Makefile:62: recipe for target 'boot/head.o' failed
make: *** [boot/head.o] Error 127
```

则下载或拷贝 `gcc-3.4` 压缩包到用户主目录并解压：

```
root@yhy:/home/yhy# ls
Desktop    Downloads  Pictures   Templates gcc-3.4.tar.gz  oslab
Documents  Music      Public     Videos   maqueos
```

```
root@yhy:/home/yhy# tar -zxvf gcc-3.4.tar.gz
```

用 `uname -m` 命令查看 `ubuntu` 版本是 32 位 or 64 位，解压后进入 `gcc-3.4` 文件夹，64 位安装 `amd64` 里的包，32 位安装 `i386` 里的包，进入对应文件夹，用 `sudo dpkg -i *.deb` 进行安装

```

root@yhy:/home/yhy# uname -m
x86_64
root@yhy:/home/yhy# ls
Desktop  Downloads  Pictures  Templates  gcc-3.4      maqueos
Documents Music      Public    Videos    gcc-3.4.tar.gz oslab
root@yhy:/home/yhy# cd gcc-3.4
root@yhy:/home/yhy/gcc-3.4# ls -l
total 8
drwxrwxr-x 2 yhy yhy 4096 Aug 18 2020 amd64
drwxrwxr-x 2 yhy yhy 4096 Aug 18 2020 i386
root@yhy:/home/yhy/gcc-3.4# cd amd64
root@yhy:/home/yhy/gcc-3.4/amd64# ls -l
total 3464
-rw-rw-r-- 1 yhy yhy 1825738 Aug 18 2020 cpp-3.4_3.4.6-8ubuntu2_amd64.deb
-rw-rw-r-- 1 yhy yhy 165122 Aug 18 2020 gcc-3.4-base_3.4.6-8ubuntu2_amd64.deb
-rw-rw-r-- 1 yhy yhy 1551014 Aug 18 2020 gcc-3.4_3.4.6-8ubuntu2_amd64.deb
root@yhy:/home/yhy/gcc-3.4/amd64# sudo dpkg -i *.deb

```

(2) 运行

在 Bochs 中运行最新编译好的内核很简单，在 oslab 目录下执行：

```

# 注意是在上层目录# 刚刚编译是在 oslab/linux-0.11/ 文件夹下
$ cd ~/oslab/
# 执行 run 脚本
$ ./run

```

如果出现 Bochs 的窗口，里面显示 linux 的引导过程，最后停止在 `[/usr/root/#]`，表示运行成功，如下图所示。

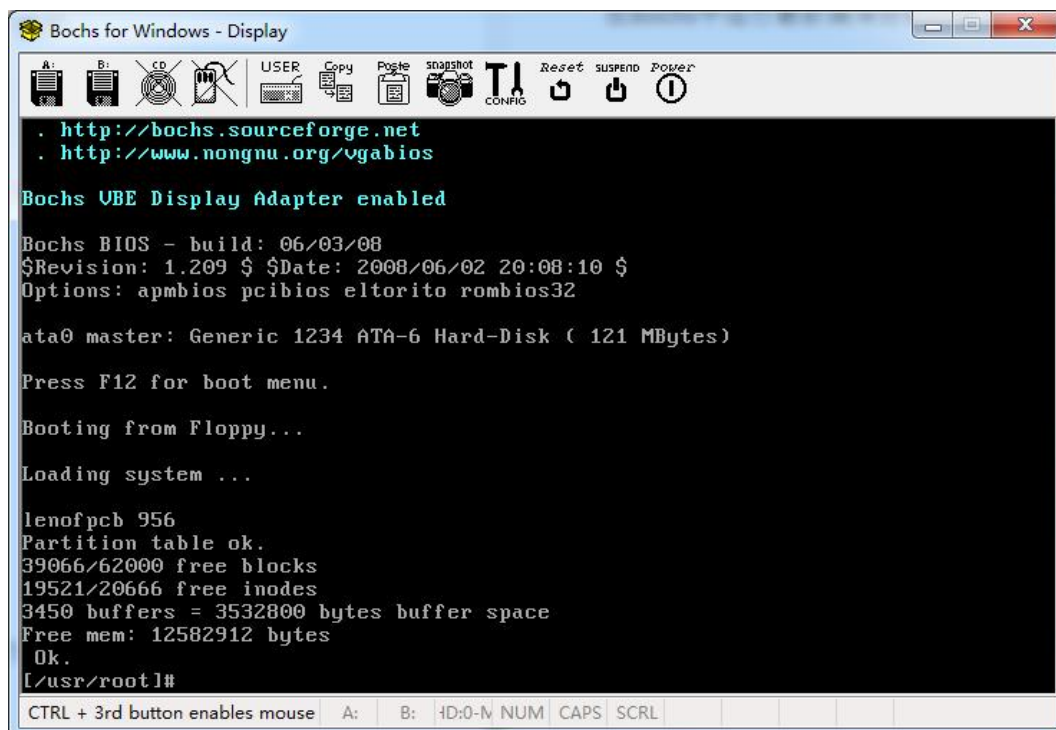


图 1 用 Bochs 启动 Linux 0.11 以后的样子

若出现如下错误：

```
root@yhy:/home/yhy/oslab# ./run
./bochs/bochs-gdb: error while loading shared libraries: libSM.so.6: cannot open
shared object file: No such file or directory
```

根据经验，还需要用 `sudo apt install` 包名，安装如下包：

```
libsm6:i386
libx11-6:i386
libxpm4:i386
```

若安装第 1 个出错，用如下命令启用 APT 对 32 位软件包的支持，然后再安装上述包

```
sudo dpkg --add-architecture i386
sudo apt-get update
```

(3) 调试

内核调试分为两种模式：汇编级调试和 C 语言级调试。

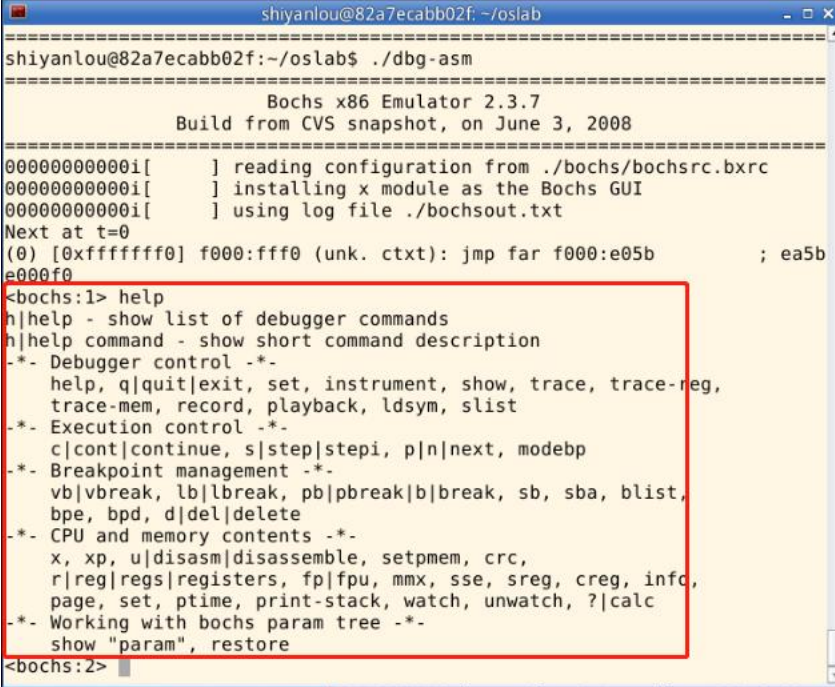
➤ 汇编级调试

汇编级调试需要执行命令：

```
# 确认在 oslab 目录下
$ cd ~/oslab/
# 运行脚本前确定已经关闭刚刚运行的 Bochs
$ ./dbg-asm
```

汇编级调试的启动之后 Bochs 是黑屏，这是正常的。

可以用命令 `help` 来查看调试系统用的基本命令。更详细的信息请查阅 Bochs 使用手册。



```
shiyanolou@82a7ecabb02f: ~/oslab
shiyanolou@82a7ecabb02f:~/oslab$ ./dbg-asm
=====
Bochs x86 Emulator 2.3.7
Build from CVS snapshot, on June 3, 2008
=====
0000000000i[      ] reading configuration from ./bochs/bochsrc.bxrc
0000000000i[      ] installing x module as the Bochs GUI
0000000000i[      ] using log file ./bochsout.txt
Next at t=0
(0) [0xffffffff] f000:fff0 (unk. ctxt): jmp far f000:e05b      ; ea5b
e000f0
<bochs:1> help
h|help - show list of debugger commands
h|help command - show short command description
--*- Debugger control -*--
    help, q|quit|exit, set, instrument, show, trace, trace-reg,
    trace-mem, record, playback, ldsym, slist
--*- Execution control -*--
    c|cont|continue, s|step|stepi, p|n|next, modebp
--*- Breakpoint management -*--
    vb|vbreak, lb|lbreak, pb|pbreak|b|break, sb, sba, blist,
    bpe, bpd, d|del|delete
--*- CPU and memory contents -*--
    x, xp, u|disasm|disassemble, setpmem, crc,
    r|reg|regs|registers, fp|fpu, mmx, sse, sreg, creg, info,
    page, set, ptime, print-stack, watch, unwatch, ?|calc
--*- Working with bochs param tree -*--
    show "param", restore
<bochs:2>
```

➤ C 语言级调试

C 语言级调试稍微复杂一些。首先执行如下命令：

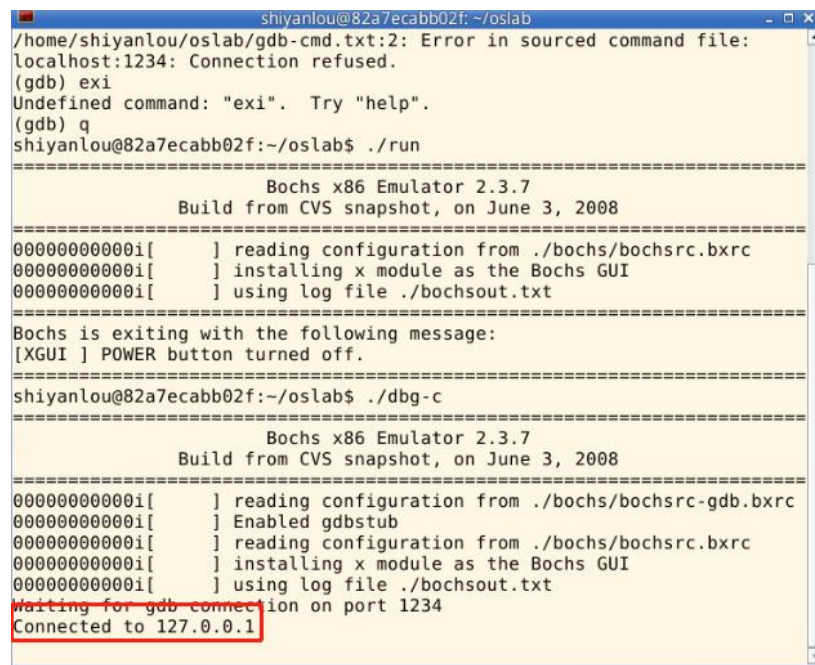
```
$ cd ~/oslab
$ ./dbg-c
```

然后再打开一个终端窗口，执行：

```
$ cd ~/oslab
$ ./rungdb
```

注意：启动的顺序不能交换，否则 gdb 无法连接。

出现下图所示的提示，才说明连接成功：



```
shiyanolou@82a7ecabb02f: ~/oslab
/home/shiyanolou/oslab/gdb-cmd.txt:2: Error in sourced command file:
localhost:1234: Connection refused.
(gdb) exi
Undefined command: "exi". Try "help".
(gdb) q
shiyanolou@82a7ecabb02f:~/oslab$ ./run
=====
Bochs x86 Emulator 2.3.7
Build from CVS snapshot, on June 3, 2008
=====
0000000000i[      ] reading configuration from ./bochs/bochsrc.bxrc
0000000000i[      ] installing x module as the Bochs GUI
0000000000i[      ] using log file ./bochsout.txt
=====
Bochs is exiting with the following message:
[XGUI ] POWER button turned off.
=====
shiyanolou@82a7ecabb02f:~/oslab$ ./dbg-c
=====
Bochs x86 Emulator 2.3.7
Build from CVS snapshot, on June 3, 2008
=====
0000000000i[      ] reading configuration from ./bochs/bochsrc-gdb.bxrc
0000000000i[      ] Enabled gdbstub
0000000000i[      ] reading configuration from ./bochs/bochsrc.bxrc
0000000000i[      ] installing x module as the Bochs GUI
0000000000i[      ] using log file ./bochsout.txt
Waiting for gdb connection on port 1234
Connected to 127.0.0.1
```

新终端窗口中运行的是 GDB 调试器。关于 gdb 调试器请查阅 GDB 使用手册。

若出现如下错误：

```
root@yhy:/home/yhy/oslab# ./rungdb
./gdb: error while loading shared libraries: libncurses.so.5: cannot open shared
object file: No such file or directory
```

则安装：

```
sudo apt install libncurses5:i386
```

```
sudo apt install libexpat1:i386
```

(4) 文件交换

接下来看 Ubuntu 和 Linux 0.11 之间的文件交换如何启动。

注：开始设置文件交换之前，务必关闭所有的 Bochs 进程。

oslab 下的 **hdc-0.11.img** 是 0.11 内核启动后的根文件系统镜像文件，相当于在 bochs 虚拟机里装载的硬盘。在 Ubuntu 上访问其内容的方法是：

```
$ cd ~/oslab/
# 启动挂载脚本
$ sudo ./mount-hdc
```

大家使用 `sudo` 时，password 是 `shiyanolou`，也有可能不会提示输入密码。之后，`hdc` 目录下就是和 0.11 内核一模一样的文件系统了，可以读写任何文件（可能有些文件要用 `sudo` 才能访问）。

```
# 进入挂载到 Ubuntu 上的目录
$ cd ~/oslab/hdc
# 查看内容
$ ls -al
```

读写完毕，不要忘了卸载这个文件系统：

```
$ cd ~/oslab/
# 卸载
$ sudo umount hdc
```

经过 `sudo ./mount-hdc` 这样处理以后，我们可以在 Ubuntu 的 `hdc` 目录下创建一个 `xxx.c` 文件，然后利用 Ubuntu 上的编辑工具（如 `gedit` 等）实现对 `xxx.c` 文件的编辑工作，在编辑保存以后。

执行 `sudo umount hdc` 后，再进入 Linux 0.11（即 `run` 启动 `bochs` 以后）就会看到这个 `xxx.c`（即如下图所示），这样就避免了在 Linux 0.11 上进行编辑 `xxx.c` 的麻烦，因为 Linux 0.11 作为一个很小的操作系统，其上的编辑工具只有 `vi`，使用起来非常不便。

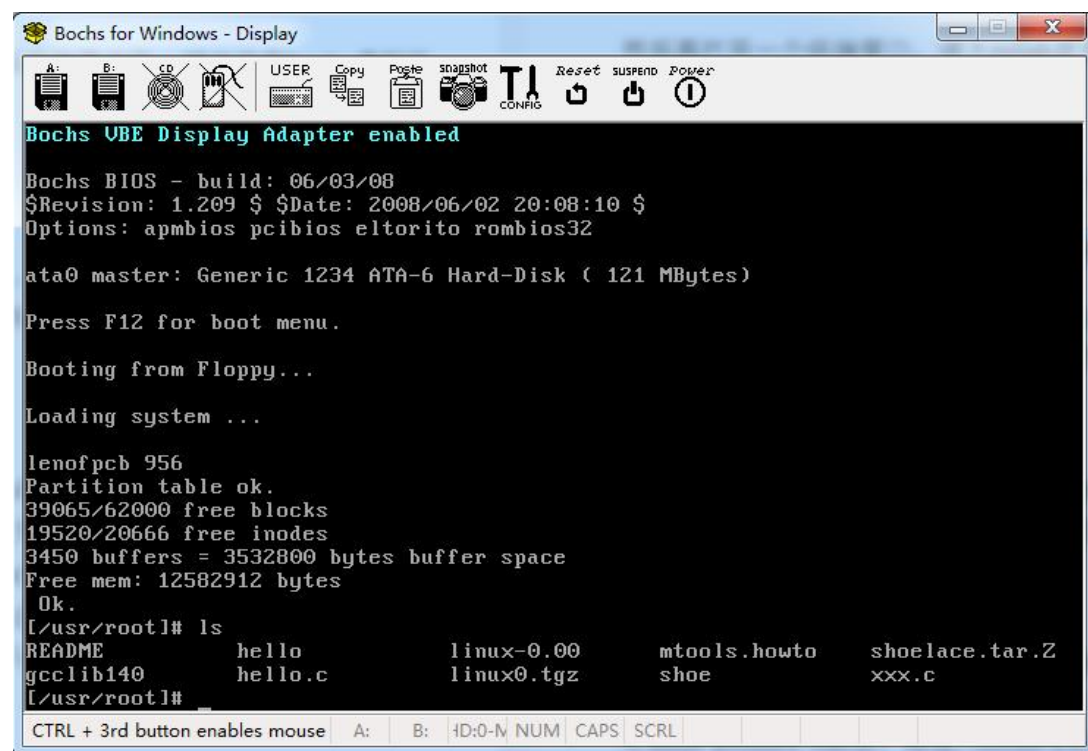


图 2 用 Ubuntu 和 Linux 0.11 完成文件交换以后再启动 Linux 0.11 以后

另外在 Linux 0.11 上产生的文件，如后面实验中产生的 `process.log` 文件，可以按这种方式“拿到”Ubuntu 下用 `python` 程序进行处理，当然这个 `python` 程序在 Linux 0.11 上显然是不好使的，因为 Linux 0.11 上搭建不了 `python` 解释环境。

注意 1：不要在 0.11 内核运行的时候 `mount` 镜像文件，否则可能会损坏文件系统。同理，也不要已经在 `mount` 的时候运行 0.11 内核。

注意 2：在关闭 `Bochs` 之前，需要先在 0.11 的命令行运行“`sync`”，确保所有缓存数据都存盘后，再关闭 `Bochs`。