**北京邮电大学课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | **操作系统课程设计** | | **学院** | **计算机学院** | **指导教师** | **叶文** |
| **班级** | **班内序号** | **学号** | | **学生姓名** | **成绩** | |
| **2014211302** | **14** | **2014211168** | | **周尧棋** |  | |
| **2014211302** | **21** | **2014211175** | | **加苏尔·吐尔地** |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | 简要介绍课程设计的主要内容，包括课程设计教学目的、基本内容、实验方法和团队分工等  主要内容：由学生分组动手进行嵌入式系统的编译和安装，熟悉嵌入式系统的开发流程，通过对linux系统源码的学习，加深对操作系统的认识。  实验内容：1、Linux启动过程优化2、Linux系统内核/系统配置小型化 3、ARM/ucLinux- SkyEye的安装、配置与集成 4、ARM/ucLinux- SkyEye下BusyBox集成——Shell小型化 5、Linux 环境下的Ramdisk技术及其实验6、Linux 环境下嵌入式数据库mSQL的集成  实验分工：周尧棋 实验1 3 4 6 加苏尔·吐尔地 实验1 2 5 | | | | | |
| **学生**  **课程设计**  **报告**  （附页） |  | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | 遵照实践教学大纲并根据以下四方面综合评定成绩：  1、课程设计目的任务明确，选题符合教学要求，份量及难易程度  2、团队分工是否恰当与合理  3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果  4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范  **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

**北京邮电大学课程设计报告**

目录

[实验一 Linux 启动优化 4](#_Toc487662830)

[一、实验目的 4](#_Toc487662831)

[二、实验环境 4](#_Toc487662832)

[三、实验内容及步骤 4](#_Toc487662833)

[3.1 安装系统 4](#_Toc487662834)

[3.2 内核态启动优化 4](#_Toc487662835)

[3.3 用户态启动优化 20](#_Toc487662836)

[四、实验总结 24](#_Toc487662837)

[实验二 Linux系统内核/系统配置小型化 24](#_Toc487662838)

[一、实验目的 24](#_Toc487662839)

[二、实验环境 25](#_Toc487662840)

[三、实验内容 25](#_Toc487662841)

[四、实验设计原理 25](#_Toc487662842)

[五、实验步骤 25](#_Toc487662843)

[六、实验结果及分析 37](#_Toc487662844)

[七、实验总结 39](#_Toc487662845)

[实验三 ARM/ucLinux-SkyEye的安装、配置与集成 40](#_Toc487662846)

[一、实验目的 40](#_Toc487662847)

[二、实验环境 40](#_Toc487662848)

[三、实验内容与步骤 40](#_Toc487662849)

[3.1 将下载的三个数据包安装并进入目录/root/skyeye 40](#_Toc487662850)

[3.2 安装skyeye 42](#_Toc487662851)

[3.3 安装arm-elf-tools交叉编译器（用于编译uclinuxskyEye） 46](#_Toc487662852)

[3.4 安装ucLinux 47](#_Toc487662853)

[3.5 配置 57](#_Toc487662854)

[3.6 运行 58](#_Toc487662855)

[3.7 测试 60](#_Toc487662856)

[3.8 运行5条常用指令 62](#_Toc487662857)

[四、实验总结 63](#_Toc487662858)

[实验四 ARM/ucLinux-SkyEye下BusyBox集成——Shell小型化 63](#_Toc487662859)

[一、实验目的 63](#_Toc487662860)

[二、实验环境 64](#_Toc487662861)

[三、实验内容与步骤 64](#_Toc487662862)

[3.1 当前shell 64](#_Toc487662863)

[3.2 busybox替换shell 66](#_Toc487662864)

[3.3 代码测试 76](#_Toc487662865)

[四、实验总结 77](#_Toc487662866)

[实验五 Linux环境下Ramdisk技术及其实验 77](#_Toc487662867)

[一、实验目的 77](#_Toc487662868)

[二、实验环境 78](#_Toc487662869)

[三、实验内容与步骤 78](#_Toc487662870)

[3.1察看内核选项CONFIG\_BLK\_DEV\_RAM，确认Linux内核支持ramdisk选项 78](#_Toc487662871)

[3.2创建/格式化1个ramdisk 设备，并安装到文件系统的相应目录结构下 79](#_Toc487662872)

[3.3运行命令“df –k /dev/ram0”，查看可使用的ramdisk空间大小 79](#_Toc487662873)

[3.4 重新修改ramdisk空间大小 80](#_Toc487662874)

[3.5 文件移动 81](#_Toc487662875)

[3.6 编写一个文件访问程序，对比将此文件分别存储于外设硬盘、ramdisk分区时，程序的执行速度。 85](#_Toc487662876)

[四、实验总结 86](#_Toc487662877)

[实验六 Linux环境下嵌入式数据库mSQL的集成 87](#_Toc487662878)

[一、实验目的 87](#_Toc487662879)

[二、实验环境 87](#_Toc487662880)

[三、实验内容与步骤 87](#_Toc487662881)

[3.1 下载安装包，在Linux环境下正确安装 87](#_Toc487662882)

[3.2 建立数据库 92](#_Toc487662883)

[3.3 利用SQL语句访问数据库系统，验证所安装系统的正确性 93](#_Toc487662884)

[3.4 参考示范程序，利用mSQL API，编写实现数据库访问功能的C语言程序。 94](#_Toc487662885)

[四、实验总结 97](#_Toc487662886)

[五、源代码 98](#_Toc487662887)

实验一 Linux 启动优化

## 一、实验目的

Linux系统从启动到登录shell界面需要花费较长时间，在普通微机上的启动过程需要十几秒或更长。如果要启动X界面，那花费的时间就更多了。启动时间过长对嵌入式系统而言，如信息家电产品（机顶盒），是无法接受的。

Linux系统的启动由内核态下的启动和用户态下的启动组成。利用本实验中采取的多种方法，可以简化Linux系统启动过程，提高Linux系统启动速度，适应嵌入式系统快速启动和实时应用的需要。

## 二、实验环境

虚拟机：VMware Workstation 12.1.1（Windows 10环境）

Linux操作系统：Red Hat Linux 9，Linux内核版本为2.4.20-8

硬件：Intel i586 PC机

## 三、实验内容及步骤

### 3.1 安装系统

在虚拟机中安装Red Hat Linux 9操作系统，在开发过程中选择安装Linux源代码开发包选项，Linux源代码在/usr/src/linux-2.4目录中。

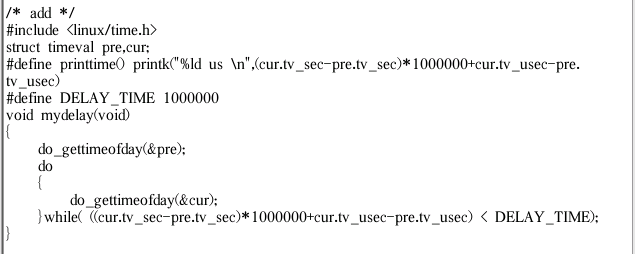
### 3.2 内核态启动优化

#### 3.2.1 测量几个较大代码模块的启动时间

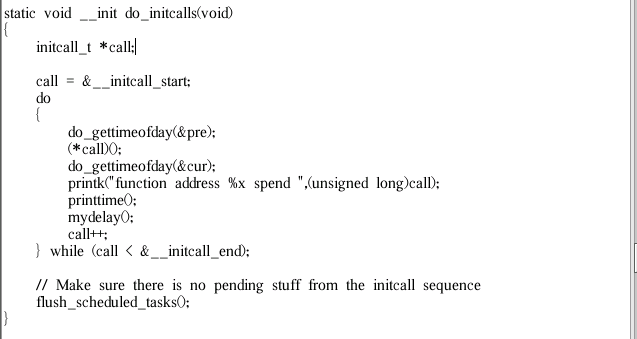
##### 3.2.1.1 修改内核代码

修改/usr/src/linux-2.4/init/main.c，添加时间测量函数，在所有需要测量的代码段添加条件编译TIME\_TEST常量，测试的代码段包括console\_init()、profile\_init()、kmem\_cache\_init()、mem\_init()等等。

时间测量函数及延时函数：

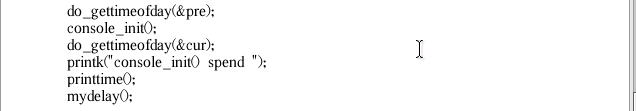


便于在启动系统时，在屏幕上打印出各个待测试模块的启动时间



在各个模块均添加时间测量函数及延时函数，打印花费时间

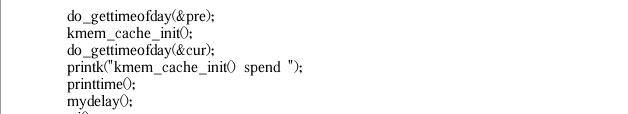
console\_init()



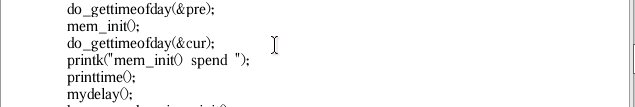
profile\_init()



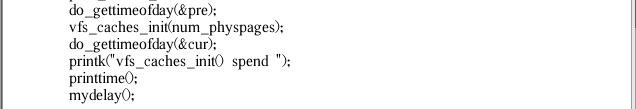
kmem\_cache\_init()



mem\_init()



vfs\_caches\_init()



buffer\_init()



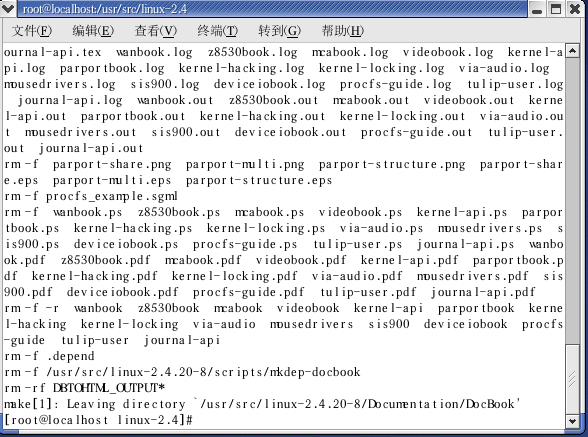
ipc\_init()



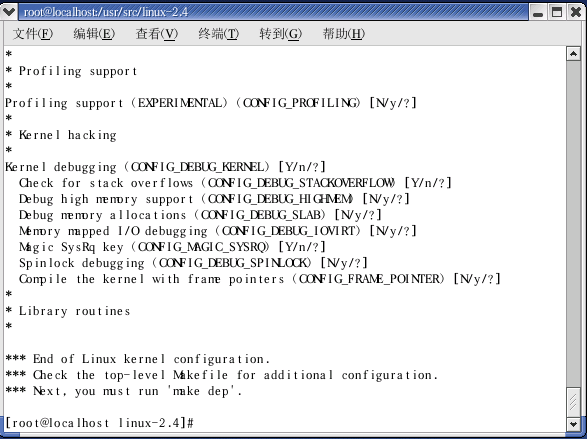
##### 3.2.1.2 编译内核

在修改完内核代码后，开始编译内核

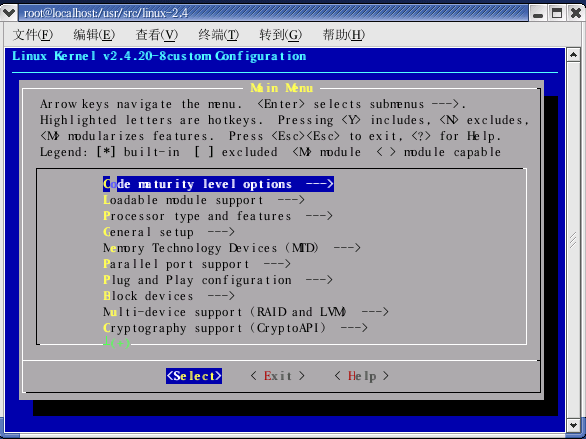
1、清除源代码：make mrproper

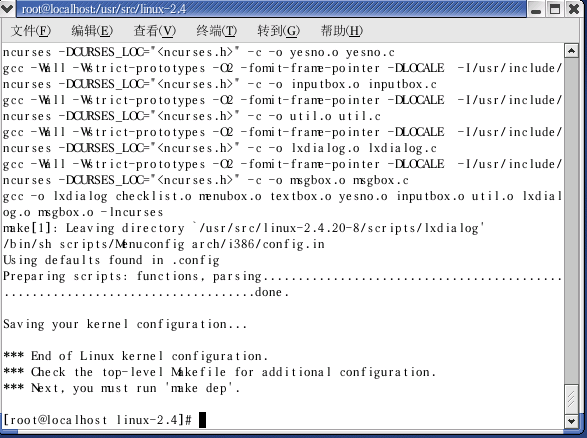


2、生成现有.config文件：make oldconfig

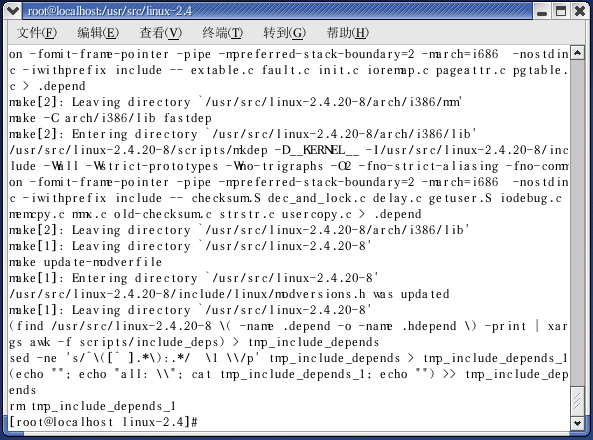


3、配置内核：make menuconfig

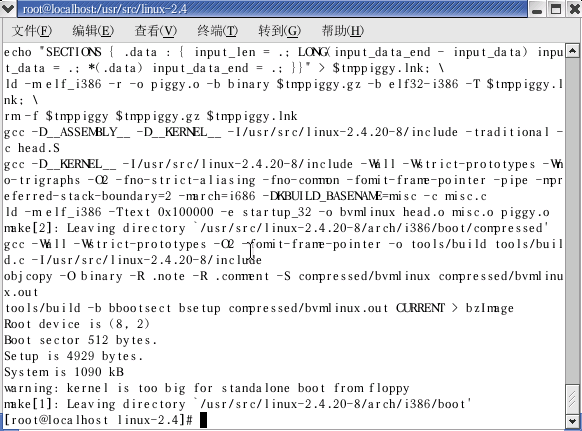




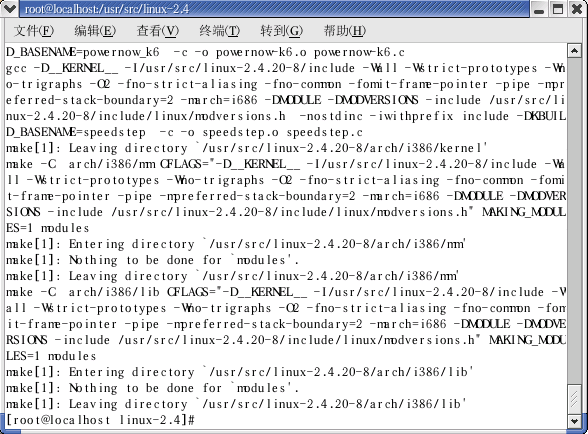
4、生成依赖关系：make dep



5、编译内核：make bzImage



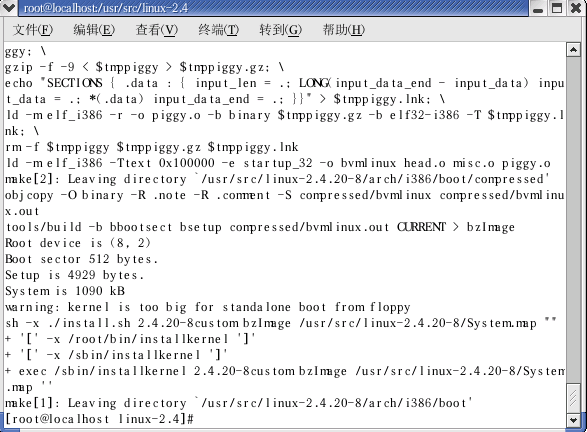
6、编译模块：make modules



7、安装模块：make modules\_install

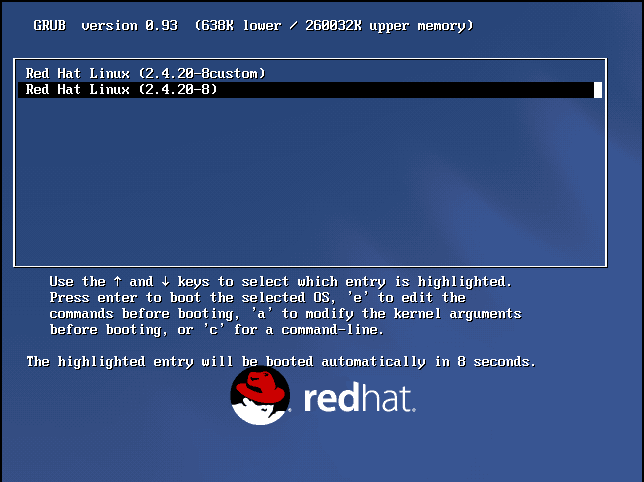


8、安装内核：make install



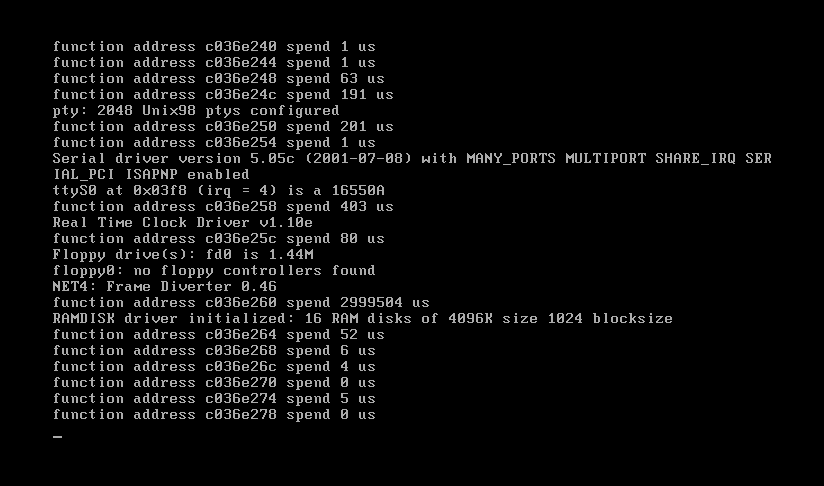
9、重启系统：reboot

重启系统之后，选择做好的系统，应为Red Hat Linux（2.4.20-8custom）那个



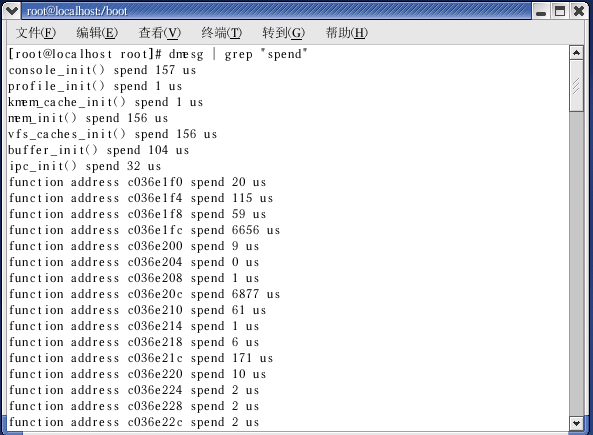
启动系统时可发现，打印出了各测试模块的启动时间

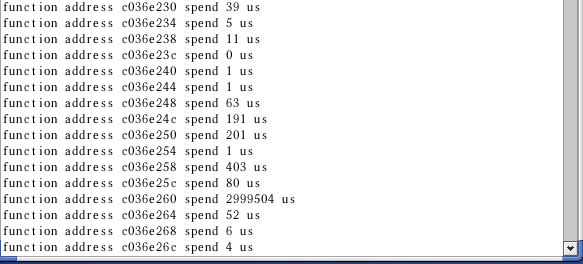


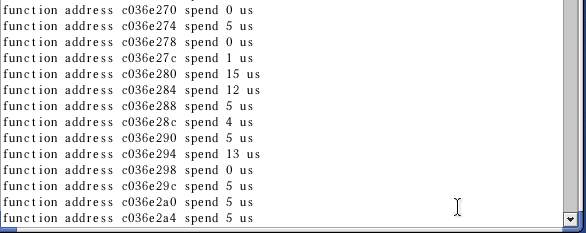


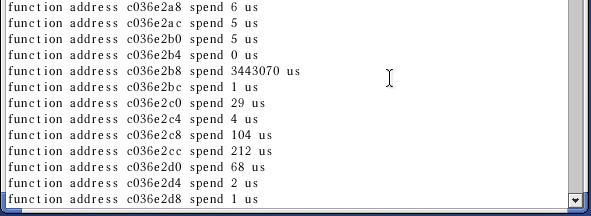
10、观察各模块启动时间

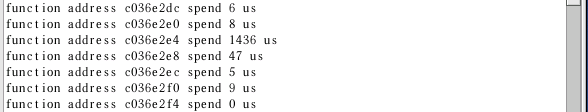
启动结束之后，用dmesg | grep "spend" 命令得到所有设定的打印位置的记录





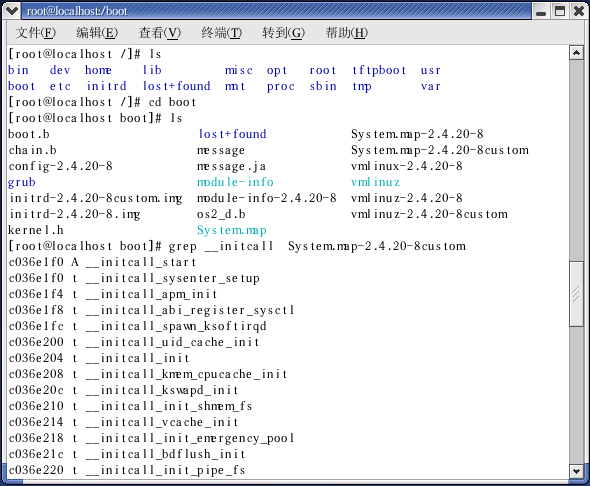


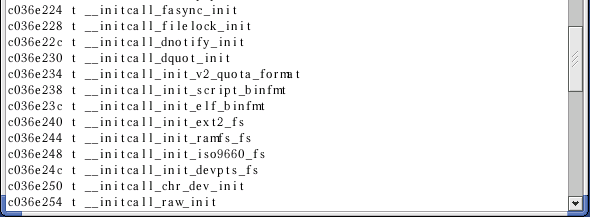


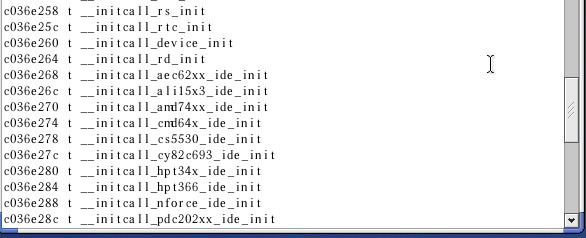


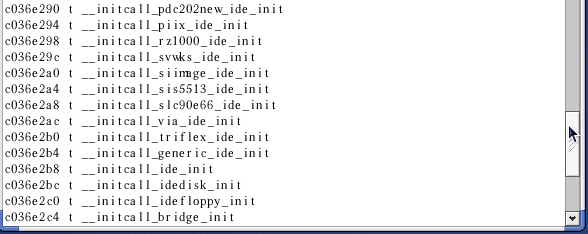
从上图可以发现，地址为c036e2b8的function花费时间最多，为3443070us。

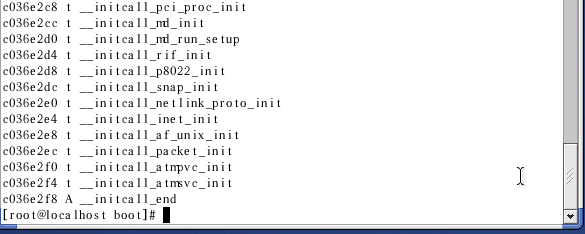
再通过查看 system.map-2.4.20-8custom文件，查出do\_initcalls() 中调用的函数地址对应的函数名：











通过分析这些数据，发现initcall\_ide\_init函数用的时间最长，为3443070us，其他函数初始化用时基本不到0.1s。

#### 3.2.2 内核态进程启动过程优化-IDE检测和修改

根据3.2.1实验，发现IDE初始化化的时间很长，下面优化这一部分

修改linux-2.4.20/include/asm-I386/ide.h中宏定义#define MAX\_HWIFS 10

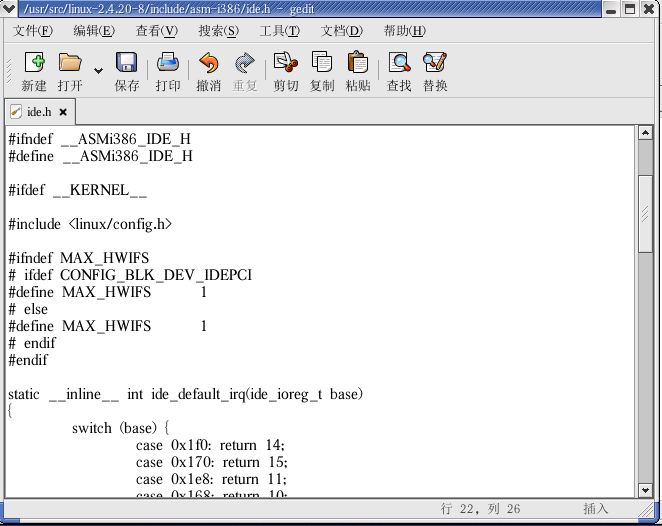
改为#define MAX\_HWIFS 3

这样开机只进行三次IDE初始化，加快启动时间。

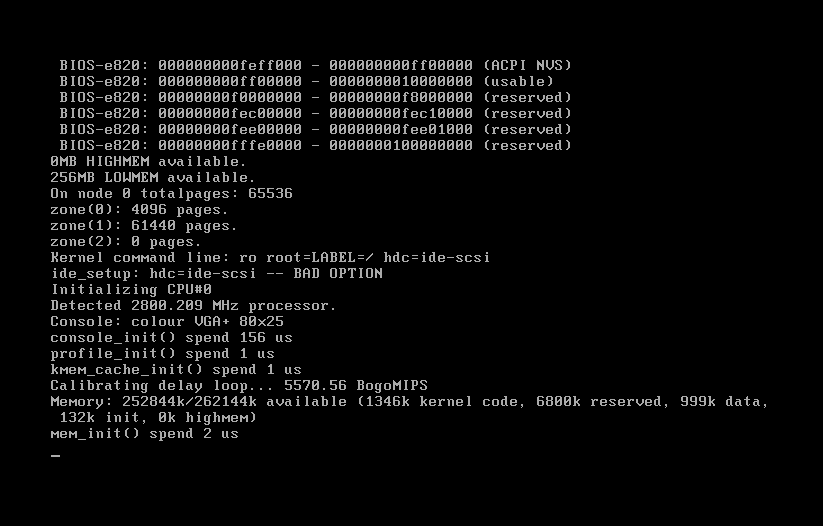
重新编译出一个新内核。

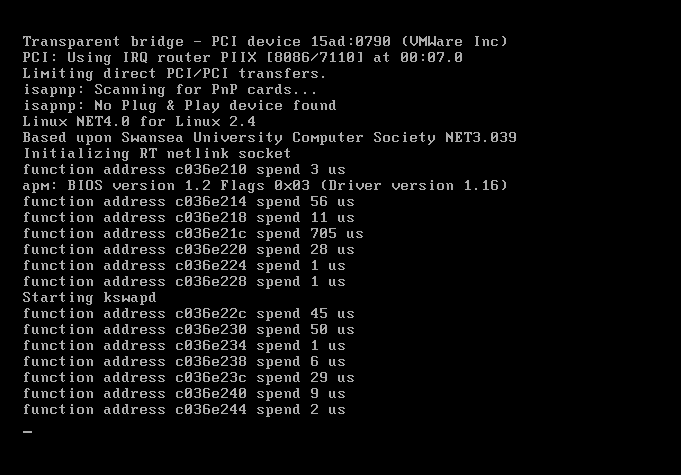
重启，使用新内核，查看IDE监测对应函数执行时间，与3.2.1实验对比，看执行速度加快了多少。

修改宏定义#define MAX\_HWIFS：

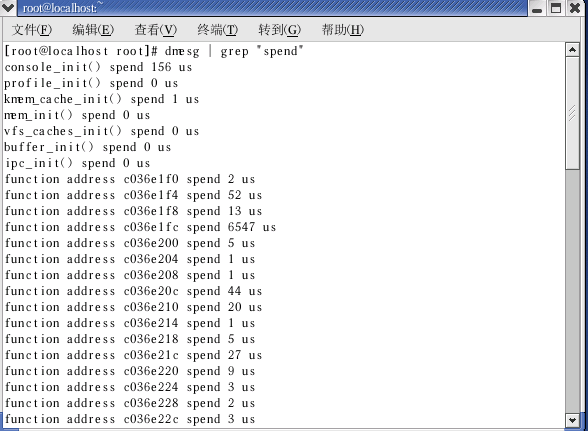


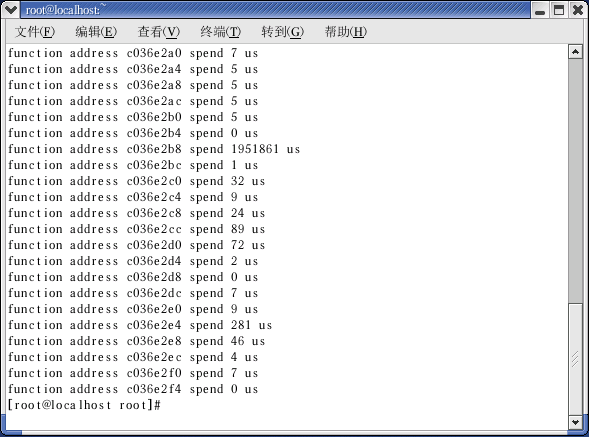
优化后启动截图：



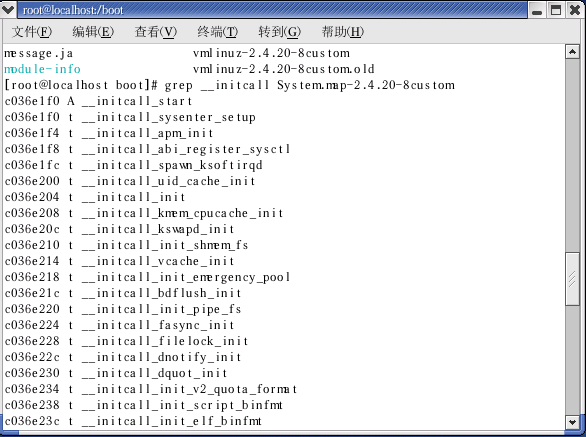


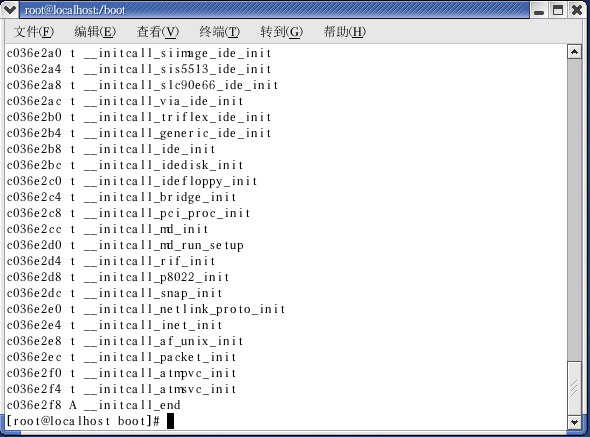
修改完之后各模块启动时间：





再通过查看 system.map-2.4.20-8custom文件，查出do\_initcalls() 中调用的函数地址对应的函数名：





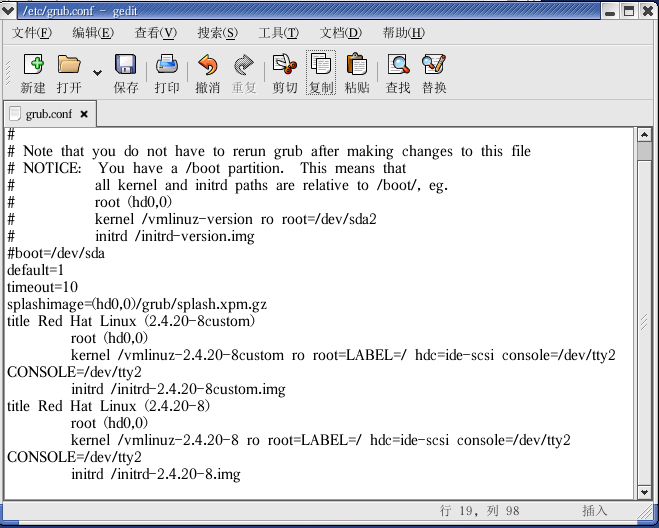
initcall\_ide\_init函数对应的地址为c036e2b8，而c036e2b8地址的函数耗费了1951861us

结合3.2.1实验，发现initcall\_ide\_init函数初始化时间从3443070us减少到了1951861us

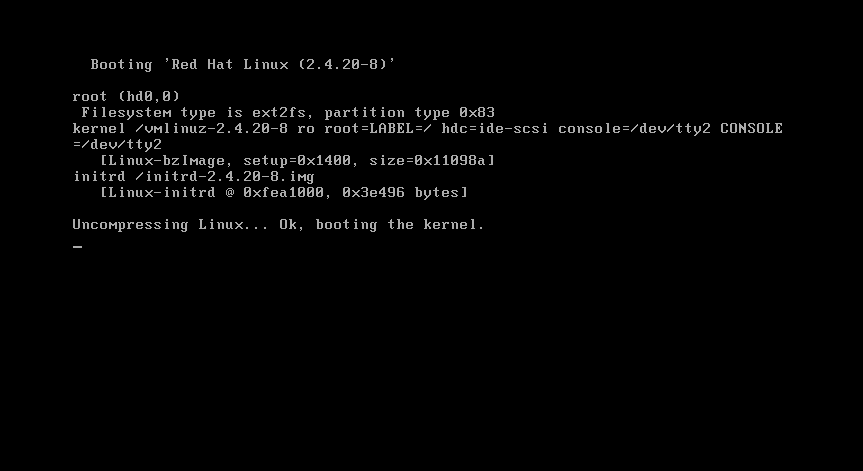
### 3.3 用户态启动优化

#### 3.3.1 开机画面字符隐藏技术

在/etc/grub.conf 中，在kernel 一行后添加 console=/dev/tty2 CONSOLE=/dev/tty2 后，重启即能隐藏内核态的输出信息。通过ALT+F2就可以调出内核态的输出信息。

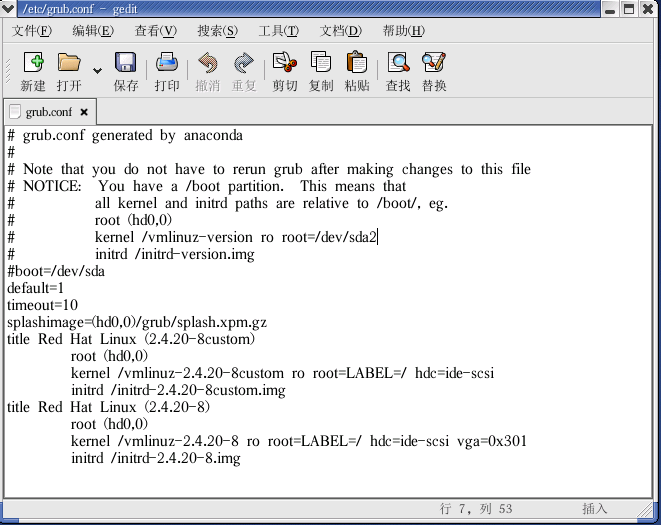


修改后，发现开机时字符已经被隐藏



#### 3.3.2 开机画面隐藏/调出

默认情况下，开机画面是系统内核打印输出，调出小企鹅通过在系统的启动参数后面加vga=0x301可以把小企鹅调出



实验结果：

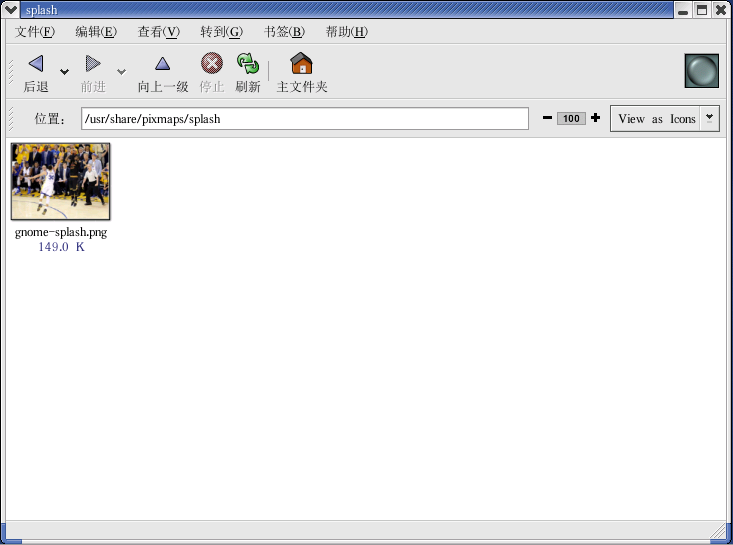


#### 3.3.3 开机画面更换

Linux系统利用底层图形基础设施Frame Buffer显示图片。Frame Buffer是一种驱动程序接口，它将显示设备抽象为帧缓冲区，因此Frame Buffer可以看作显存的一个映像，映射到地址空间。

启动画面是指系统每次进入桌面环境时出现的小幅画面，它表明系统正在加载程序。在 GNOME环境下修改/替换启动画面的原理为：用一幅大小适中、文件格式为PNG的图片，假定图片名称为modified.png，用下述命令替换掉/usr/share/pixmaps/splash/gnome-splash.png

cp modified.png /usr/share/pixmaps/splash/gnome-splash.png



原来的开机启动画面：



修改完图片之后的开机画面：



## 四、实验总结

通过这次学习，我深入了解了操作系统内核，对于Linux启动的整个流程有了更深的理解，通过测试各个模块的启动时间优化启动让我对内核代码更加熟悉。

# 实验二 Linux系统内核/系统配置小型化

## 一、实验目的

1个Linux系统发行版本可多达上百兆，功能众多，支持众多硬件设施。但在一个实际Linux应用系统中，并非用到Linux系统发行版本所提供的全部功能。所以，可以针对实际计算机系统的硬件配置及其应用需求，对Linux系统内核/系统进行合理的选择和裁减，得到一个更接近实际需要的、无冗余、启动和运行更为高效的Linux系统。

例如，嵌入式系统存储容量有限、支持的硬件外设类型也有限，因此可通过内核/系统配置与裁减，选择嵌入式应用系统所必需的一些内核/系统功能（如设备驱动程序）进行配置。由此得到一个满足系统功能、体积更小的、可放入嵌入式系统的容量较小Flash中的Linux系统内核。

本实验要求学生针对PC机上的Linux系统发行版本，利用make menuconfig方法，在对硬件深入了解的基础上，选择相应的设备驱动程序和与主机CPU相关的Linux部分，进行Linux系统内核/系统配置，掌握Linux系统内核/系统的配置方法。

## 二、实验环境

硬件：Intel i586 PC机

软件：Red Hat Linux 9 (Linux内核版本为2.4.20-8)

## 三、实验内容

利用makemenuconfig方法,将不必要的内核功能去掉,只留下最基本的一些功能模块,如CPU为i586、IDE 设备驱动、TCP/IP协议栈、ne2000网卡驱动、Minix文件系统等,完成内核配置工作。

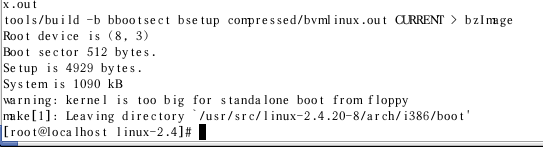
比较配置前后Linux内核/系统的体积大小、功能差异,指明配置过程中各配置选项的作用

## 四、实验设计原理

根据make menuconfig中的菜单项配置Linux内核

## 五、实验步骤

开始时使用make bzImage命令查看内核大小



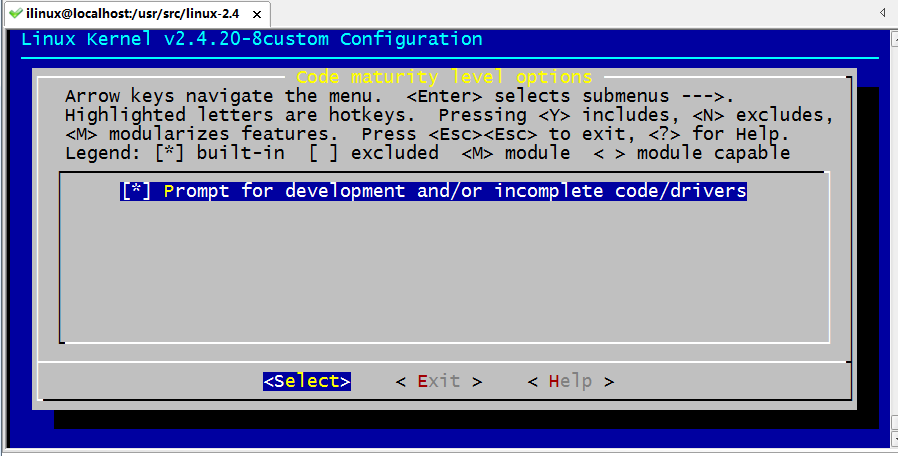
查看内核大小

使用make menuconfig配置内核，为了最小化内核，我们将一些不必要的功能都去掉。

各功能简介：

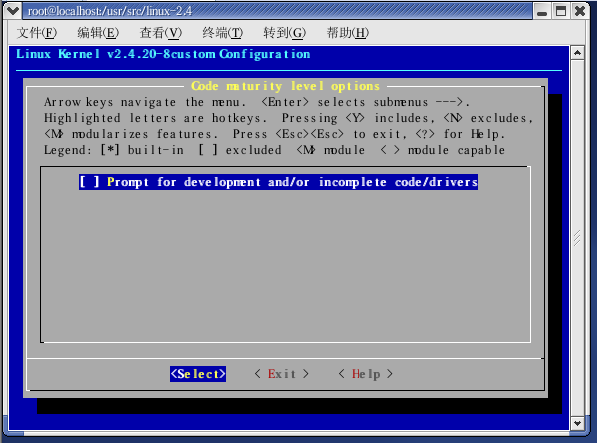
**Code maturity level options（代码成熟度选项）**

代码成熟等级。此处只有一项：prompt for development and/or incomplete code/drivers，如果你要试验现在仍处于实验阶段的功能，比如khttpd、IPv6等，就必须把该项选择为Y了；否则可以把它选择为N。



代码成熟等级

**Processor type and features（****处理器类型和特色）**

High Memory Support：大容量内存的支持。可以支持到4G、64G，一般可以不选。

Math emulation：协处理器仿真。协处理器是在386时代的宠儿，现在早已不用了。

MTTR （memory type range register）support------选择该选项，系统将生成/proc/mtrr文件对MTRR进行管理，供X server使用。同时用来启动pentinum pro和pentinum II 的特殊功能，如果你用的不是这类CPU就选N，否则也仅仅是使内核变大而已。

Symmetric multi-processing support：对称多处理支持。除非你富到有多个CPU，否则就不用选了



**处理器类型和特色**

**Loadable module support（****可加载模块支持）**

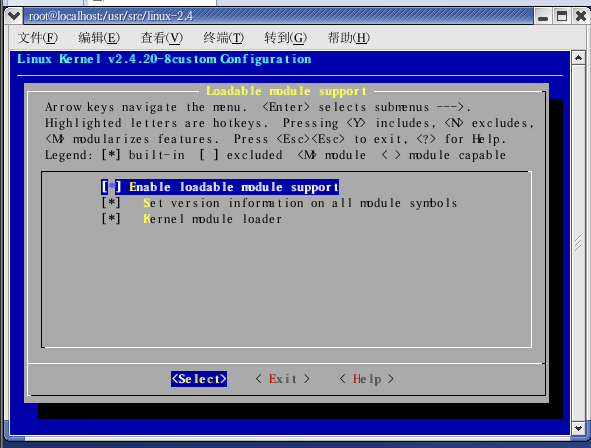
Enable loadable module support：除非你准备把所有需要的内容都编译到内核里面，否则该项应该是必选的。

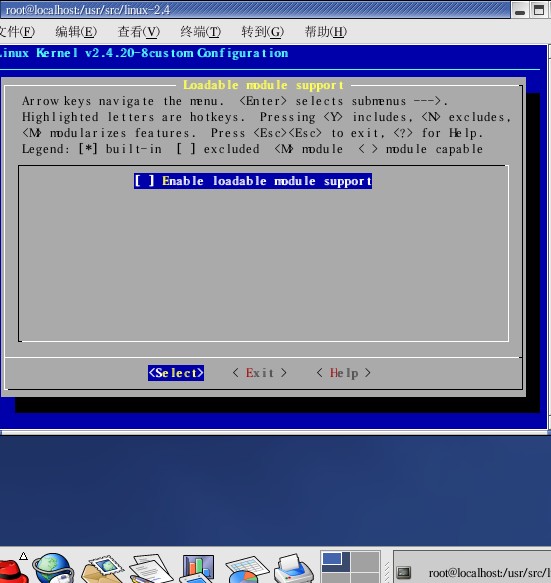
Set version inFORMation on all module symbols：可以不选它。

Kernel module loader：让内核在启动时有自己装入必需模块的能力，建议选上。   
Enable loadable module support (CONFIG\_MODULES) [Y/n/?] 选择内核是否支持加载模块。

Kernel module loader (CONFIG\_KMOD) [N/y/?] 内核将自动加载那些可加载模块，否则需要用户手工加载。

为了简化系统，在这个实验里不选择可加载内核模块的支持，选择N.





**可加载模块支持**

**General setup（****一般设置）**

一般使用缺省设置就可以了。

Networking support：网络支持。必须，没有网卡也建议你选上。

PCI support：PCI支持。如果使用了PCI的卡，当然必选。

PCI access mode：PCI存取模式。可供选择的有BIOS、Direct和Any，选Any吧。

Support for hot-pluggabel devices：热插拔设备支持。

PCMCIA/CardBus support：PCMCIA/CardBus支持。

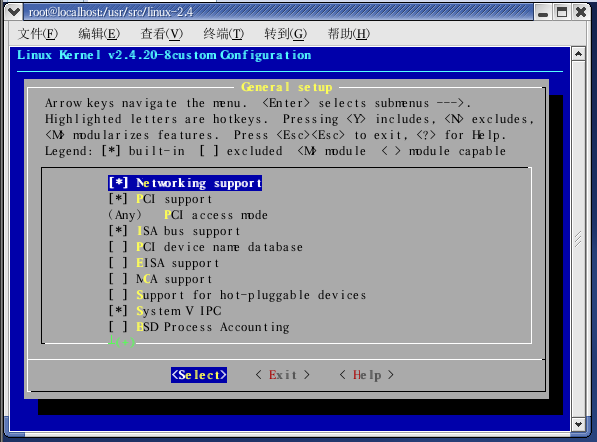
System V IPC

BSD Process Accounting

Sysctl support：以上三项是有关进程处理/IPC调用的，主要就是System V和BSD两种风格。如果你不是使用BSD，就按照缺省吧。

Power Management support：电源管理支持。

Advanced Power Management BIOS support：高级电源管理BIOS支持。



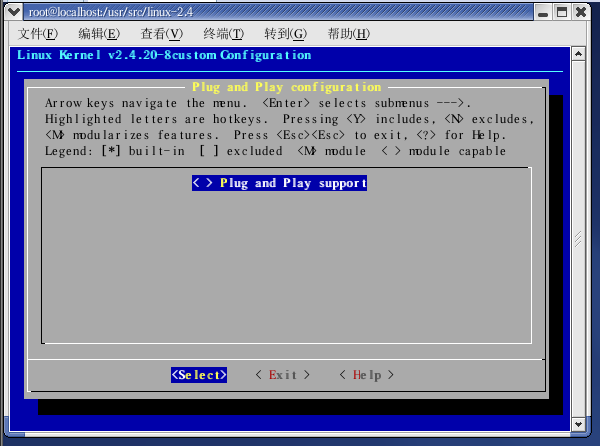
**一般设置**

**Plug and Play configuration（****即插即用设备支持）**

Plug and Play support (CONFIG\_PNP) [Y/m/n/?]内核将自动配置即插即用设备。

ISA Plug and Play support (CONFIG\_ISAPNP) [Y/m/n/?]内核将自动配置基于ISA总线的即插即用设备。

我们不需要配置，全不选。



**即插即用设备支持**

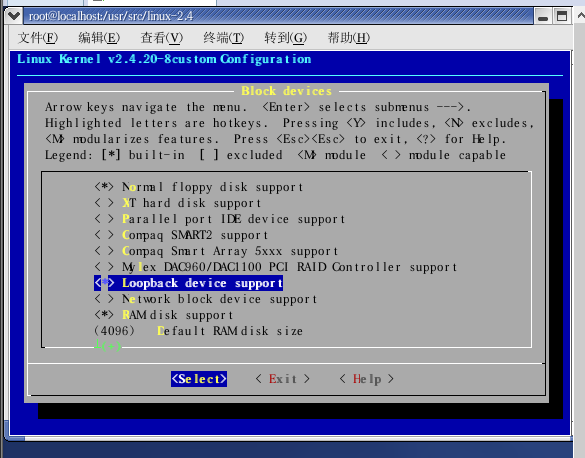
**Block devices（****块设备）**

Normal PC floppy disk support：普通PC软盘支持。这个应该必选。

Mylex DAC960/DAC1100 PCI RAID Controller support：RAID镜像用的。

Network block device support：网络块设备支持。如果想访问网上邻居的东西，就选上。

Loopback device support必选 否则后来在新内核中会无法编译新内核



**块设备**

**Networking options（****网络选项）**

Packet socket (CONFIG\_PACKET) [Y/m/n/?] 选择是否使一些应用程序使用Packet协议直接同网络设备通讯，而不通过内核中的其它中介协议。

Network firewalls (CONFIG\_FIREWALL) [N/y/?] 选择内核是否将支持防火墙。

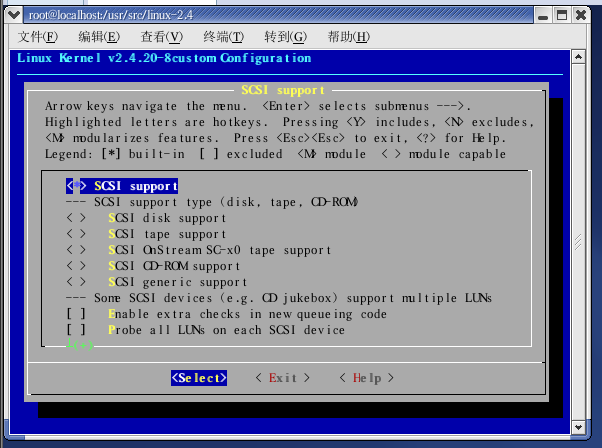
TCP/IP networking (CONFIG\_INET) [Y/n/?] 内核是否支持TCP/IP协议。

The IPX protocol (CONFIG\_IPX) [N/y/m/?] 选择内核是否支持IPX协议。

Appletalk DDP (CONFIG\_ATALK) [N/y/m/?] 选择内核是否支持Appletalk DDP协议。 

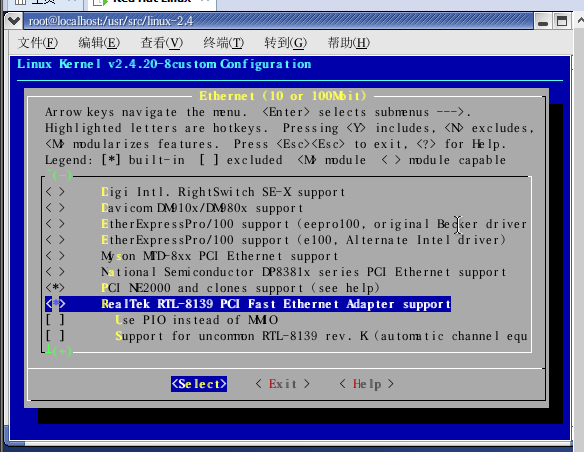
**网络选项**

**SCSI support（****SCSI支持）**

如果用户要使用SCSI设备，可配置相应选项。 由于vmware是基于scsi硬盘所以我们需要。

**SCSI支持**

**Network device support（****网络设备支持）**

Network device support (CONFIG\_NETDEVICES) [Y/n/?] 选择内核是否提供对网络驱动程序的支持。

**网络设备支持**

在Ethernet (10 or 100Mbit)（10M或100M以太网） 设置中，系统提供了许多网卡驱动程序，我们只要选择自己的网卡驱动就可以了。

**Character devices（****字符设备）**

（1）Virtual terminal (CONFIG\_VT) [Y/n/?] 选择内核是否支持虚拟终端。

（2）Support for console on virtual terminal (CONFIG\_VT\_CONSOLE) [Y/n/?] 选择内核是否将一个虚拟终端用作系统控制台。

（3）Standard/generic (dumb) serial support (CONFIG\_SERIAL) [Y/m/n/?] 选择内核是否支持串行口。

（4）Support for console on serial port (CONFIG\_SERIAL\_CONSOLE) [N/y/?] 选择内核是否将一个串行口用作系统控制台。

这里我们需要选择（1）和（2）。



**字符设备**

**Filesystems（****文件系统）**

（1）Quota support (CONFIG\_QUOTA) [N/y/?] 选择内核是否支持磁盘限额。

（2）Kernel automounter support (CONFIG\_AUTOFS\_FS) [Y/m/n/?] 选择内核是否提供对automounter的支持，使系统在启动时自动 mount远程文件系统。

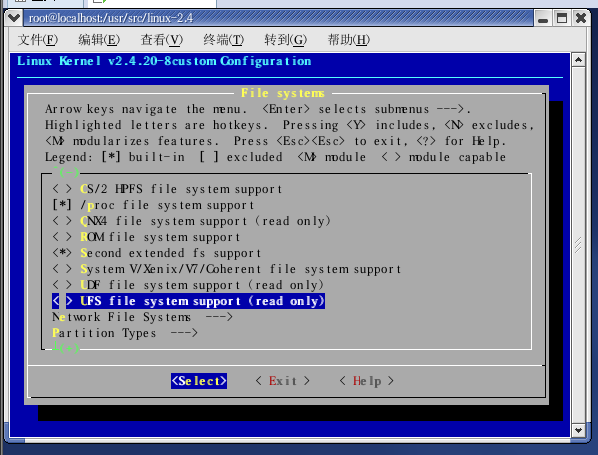
（3）DOS FAT fs support (CONFIG\_FAT\_FS) [N/y/m/?] 选择内核是否支持DOS FAT文件系统。

（4）ISO 9660 CDROM filesystem support (CONFIG\_ISO9660\_FS) [Y/m/n/?] 选择内核是否支持ISO 9660 CDROM文件系统。

（5）NTFS filesystem support (read only) (CONFIG\_NTFS\_FS) [N/y/m/?] 选择用户是否以只读方式访问NTFS文件系统。

（6）/proc filesystem support (CONFIG\_PROC\_FS) [Y/n/?] /proc是存放Linux系统运行状态的虚拟文件系统，该项必须选择“y”。

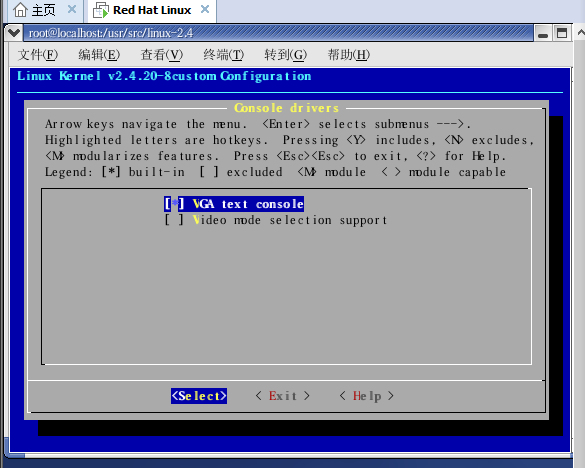
（7）Second extended fs support (CONFIG\_EXT2\_FS) [Y/m/n/?] EXT2是Linux的标准文件系统，该项也必须选择“y”。



**文件系统**

**Console drivers（****控制台驱动）**

VGA text console (CONFIG\_VGA\_CONSOLE) [Y/n/?] 选择用户是否在标准的VGA显示方式下使用Linux了。

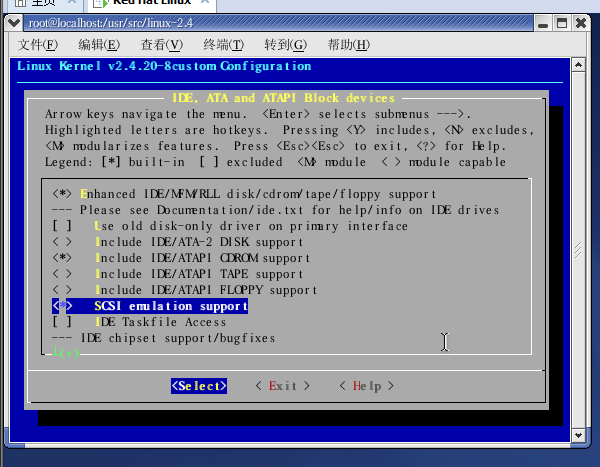


**控制台驱动**

**ATA/IDE/MFM/RLL support**

这个需要选上，IDE设备驱动还是要有的，除非你不接硬盘，把ATA/IDE/MFM/RLL

support选上后，IDE,ATA and ATAP1 Block devices就可以进入了.。



**控制台驱动**

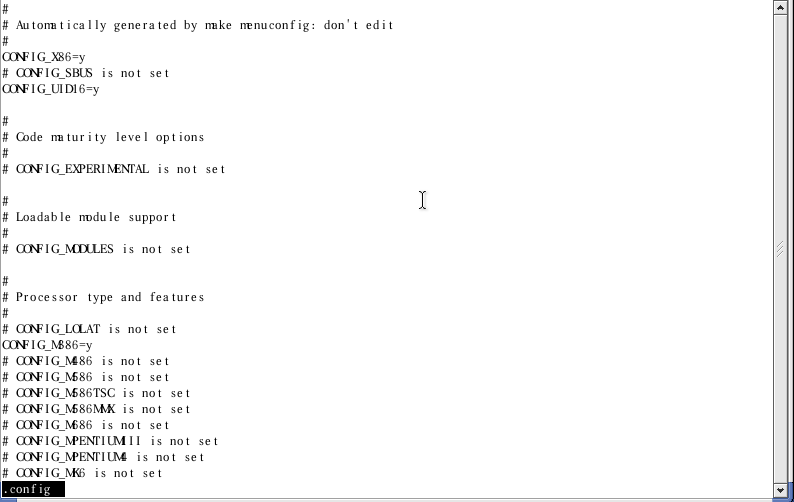
Sound（声音）

Sound card support (CONFIG\_SOUND) [N/y/m/?] 选择内核是否提供对声卡的支持。我们不需要选择。 还有Memory Technology Device Support，Parallel port support，Multiple device support，Telephy support，Fusion MPT device support，IEEE 1394(FireWire) support，I2O dexice support ，Amateur Radio support业余无线电支持、IrDA(infrafed) support红外线支持、ISDN subsystem 电话支持、Old CD-ROM drivers老的CD-ROM设备支持、Input core support，USB support、Bluetooth support、kernel hacking都不必要。

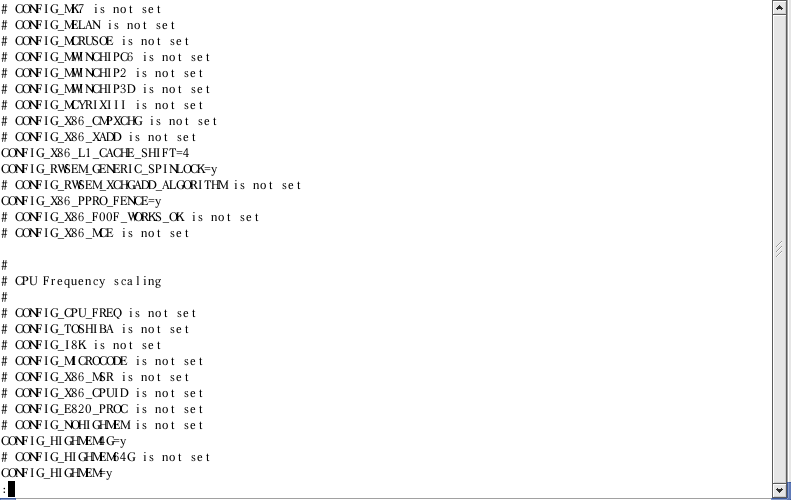
## 六、实验结果及分析

查看文件中的配置参数：

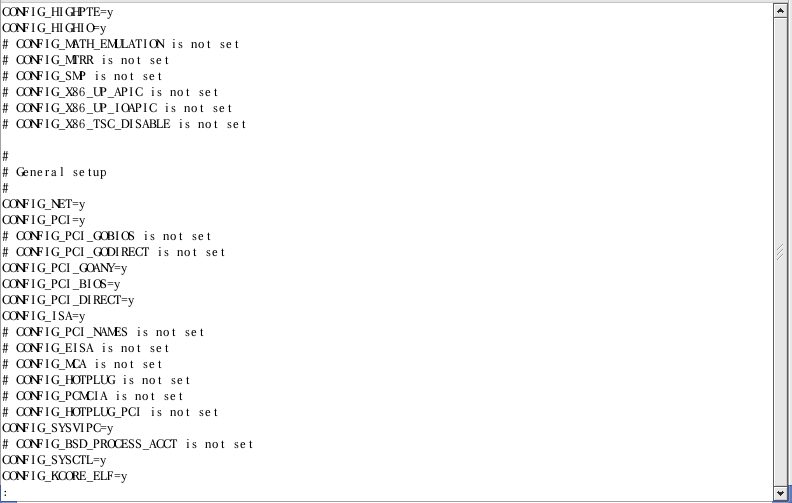
下面为.config文件的一部分



配置参数



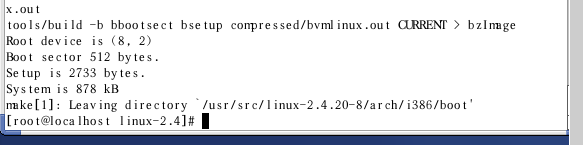
配置参数



配置参数

使用命令：make dep 生成依赖关系；make bzImage 生成系统镜像。

查看内核：



查看内核

发现内核明显减小了。

## 七、实验总结

这次实验让我更加熟悉了有关于linux内核有关问题。在实验操作中一点一点缩减内核，使得自己的内核缩减成功。虽然整个实验中虽然遇到了很多问题，但是在客服问题的过程中更加深刻地明白了linux的可用性和自由度。这次实验让我受益匪浅。

# 实验三 ARM/ucLinux-SkyEye的安装、配置与集成

## 一、实验目的

安装SkyEye嵌入式硬件模拟器，配置嵌入式操作系统ucLinux，建立基于SkyEye 的ARM/ucLinux嵌入式系统平台。

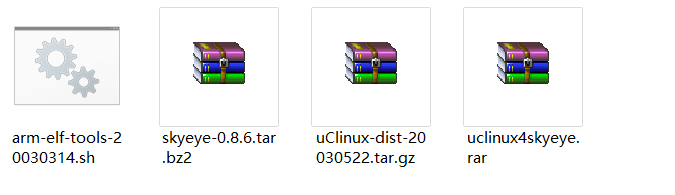
## 二、实验环境

虚拟机：VMware Workstation 12.1.1（Windows 10环境）

Linux操作系统：Red Hat Linux 9，Linux内核版本为2.4.20-8

硬件：Intel i586 PC机

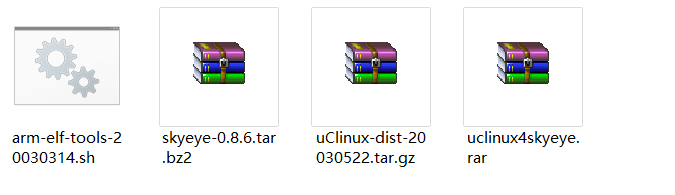
SkyEye，ucLinux ，交叉编译器，uclinuxskyeye版本如下：



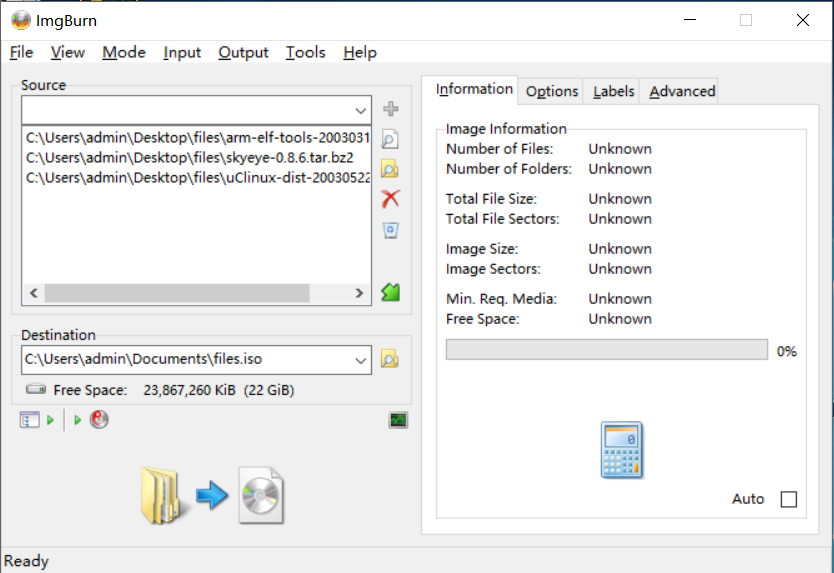
## 三、实验内容与步骤

### 3.1 将下载的三个数据包安装并进入目录/root/skyeye

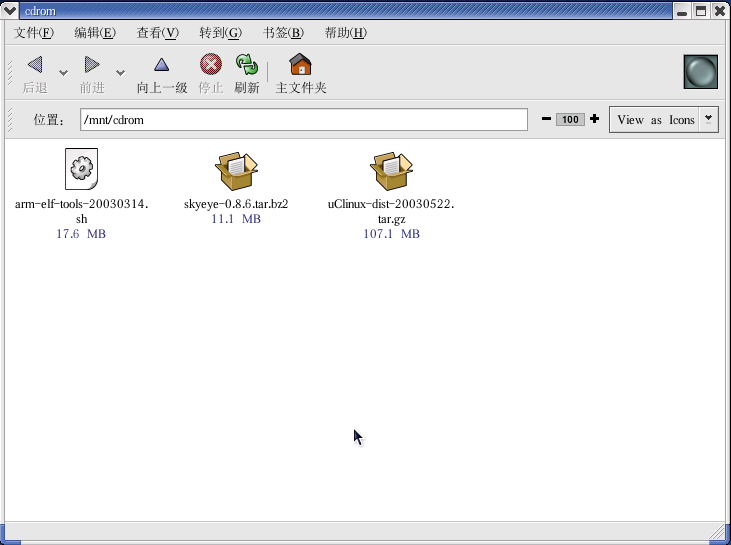
首先，Red Hat中如果不下软件不能进行主机和虚拟机之间的文件共享和拖动，所以我们使用iso文件来传输

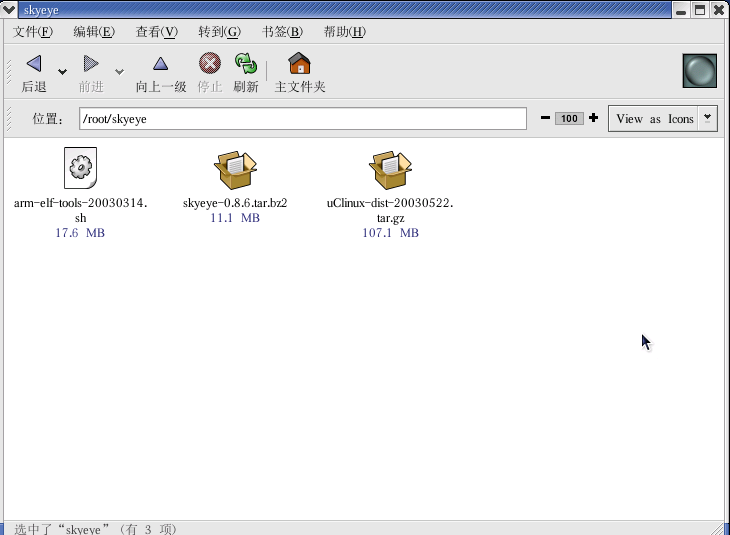


将以上文件使用软件ImgBurn转化成.iso文件



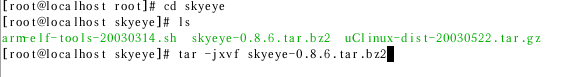
导入Linux系统后，将其复制到/root/skyeye文件夹下



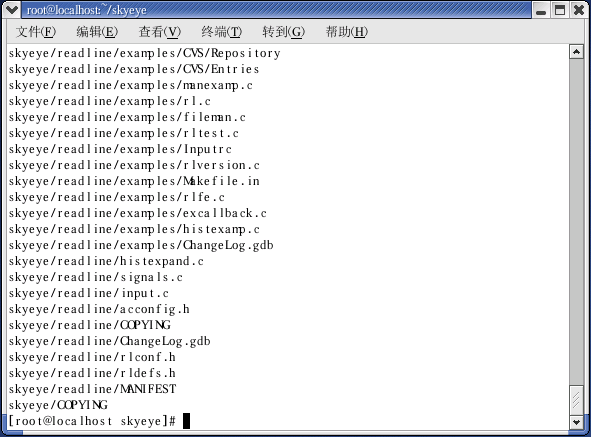


### 3.2 安装skyeye

#### 3.2.1 解压skyeye：tar -jxvf skyeye-0.8.6.tar.bz2



解压完成：

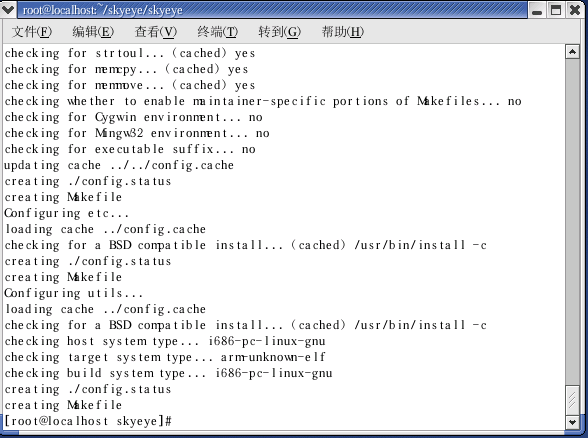


#### 3.2.2 进入skyeye：cd skyeye

#### 3.2.3预处理：./configure --target=arm-elf --prefix=/usr/local

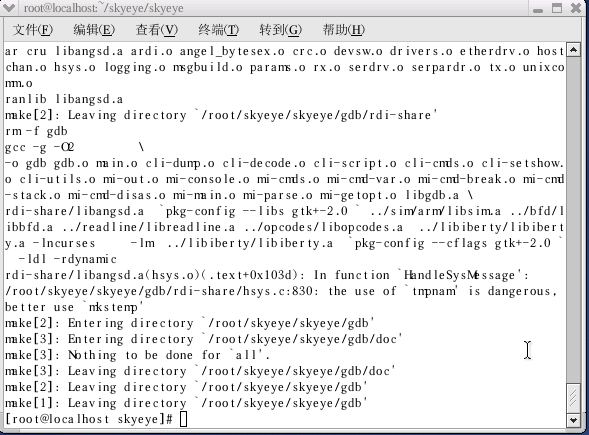


执行结果：

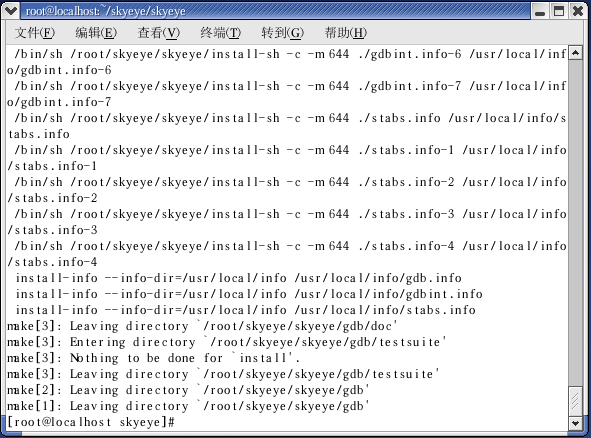


#### 3.2.4 编译安装：make和make install

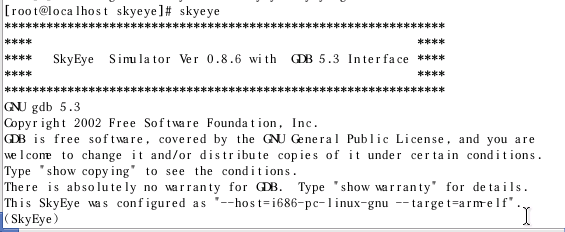
make结果：



make install结果：

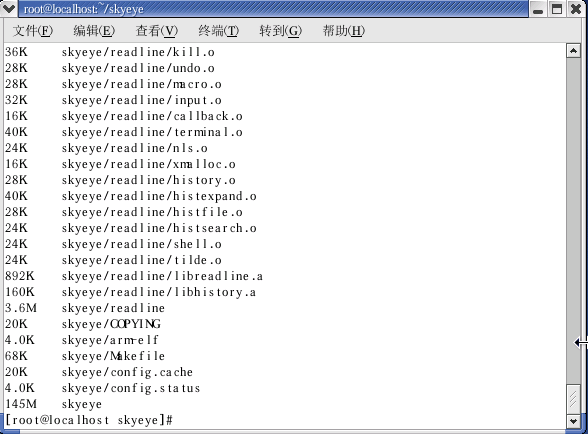


安装成功，执行命令skyeye进入skyeye状态检测：



执行du -ah skyeye，查看代码量





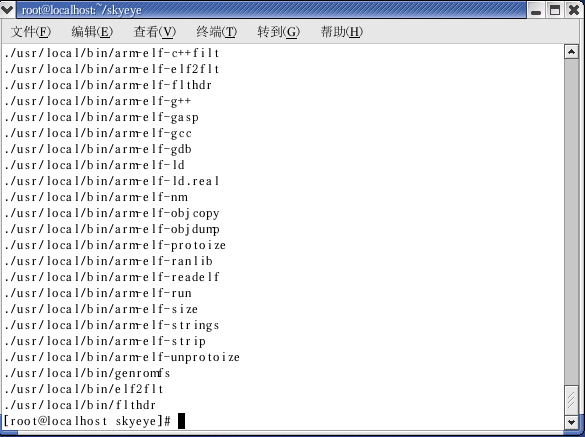
### 3.3 安装arm-elf-tools交叉编译器（用于编译uclinuxskyEye）

#### 3.3.1 使用root用户登录，以避免arm-elf-tools-20030314.sh没有执行权限的问题

#### 3.3.2 安装：./ arm-elf-tools-20030314.sh

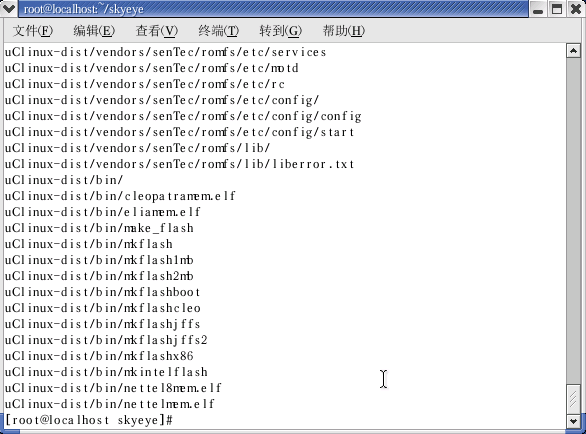


安装完毕：

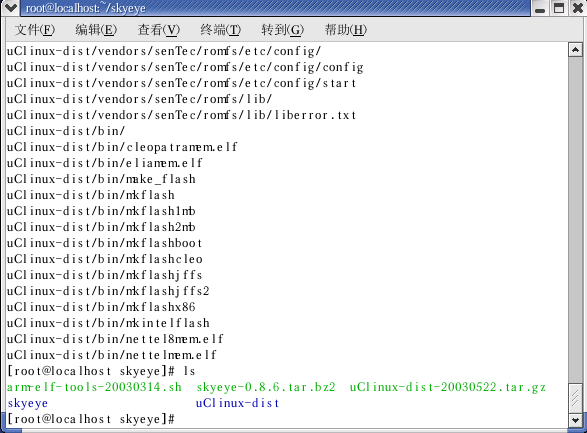


### 3.4 安装ucLinux

#### 3.4.1 解压：tar -zxvf uClinux-dist-20030522.tar.gz



解压完成，执行ls查看生成的目录：



skyeye和uCliunx-dist两个文件均已经生成

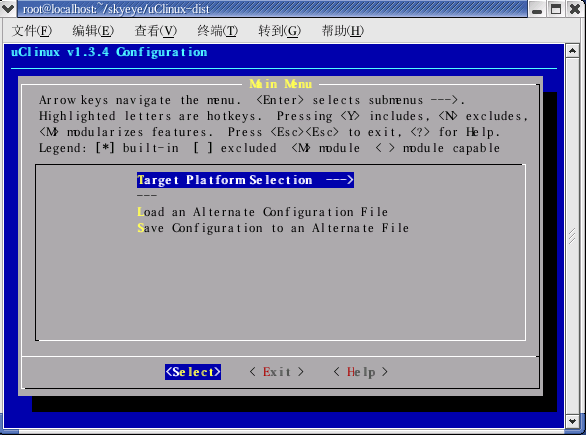
#### 3.4.2 进入cd uClinux-dist

#### 3.4.3 执行make menuconfig



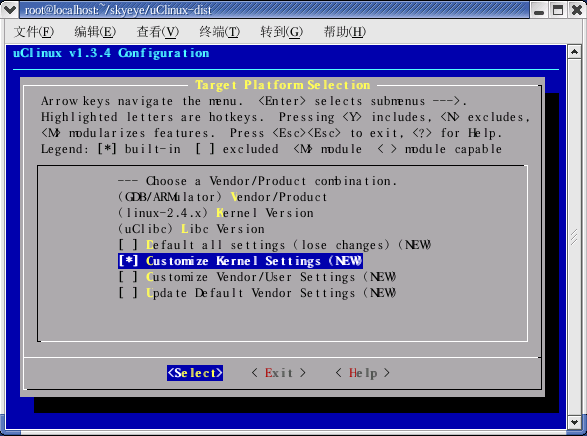
#### 3.4.4 选择模拟平台：

Target Platform Selection -> Vendor/Product -> GDB/ARMulator





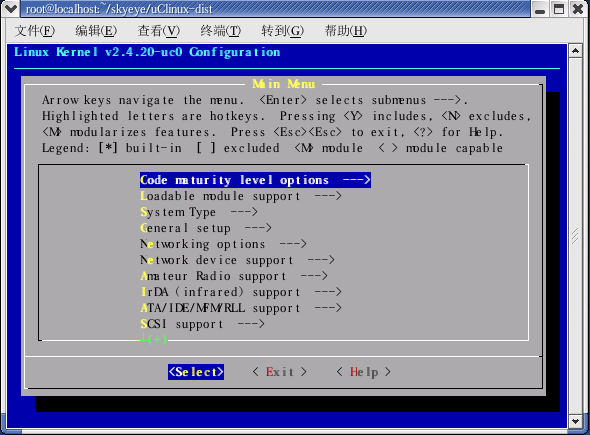
选择Customize Kernel Settings

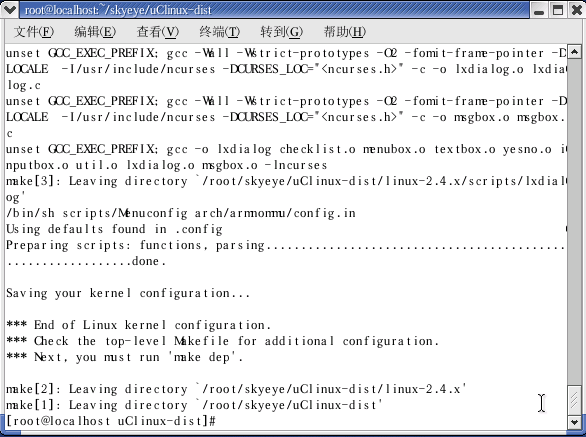


#### 3.4.5 Exit退出保存

#### 3.4.6 继续配置uClinux以支持网络

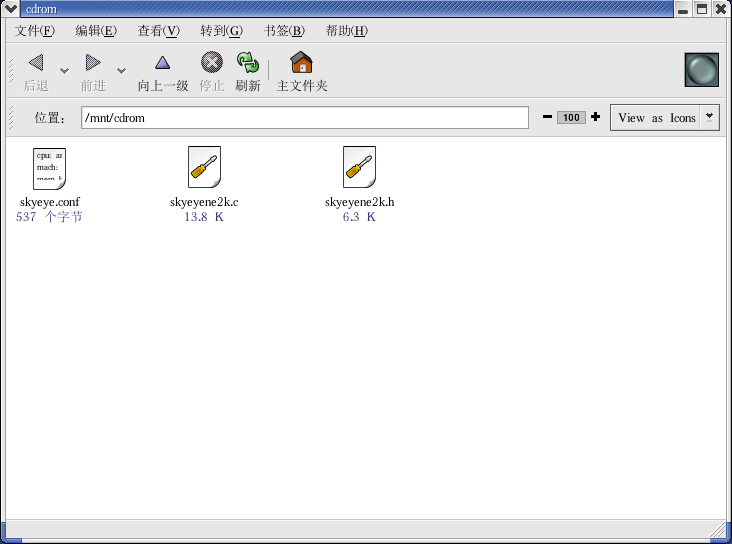
选择Network device support -> Ethernet(10 or 100Mbit) -> SkyEye ethernet support(for ARMUlator)

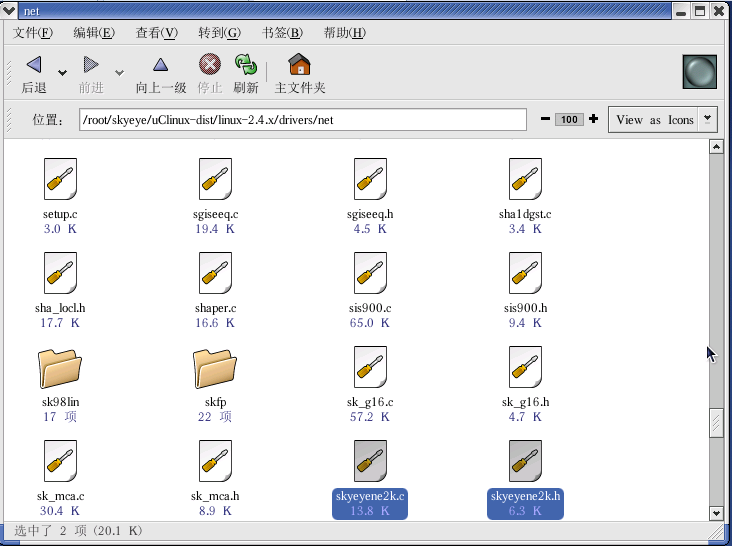




但是此时发现并没有SkyEye ethernet support(for ARMUlator)这个选项，在网上查询资料后发现了解决方法：

（1）在网上下载skyeyene2k.c和skyeyene2k.h两个文件，通过前面的iso导入法导入文件夹/root/skyeye/uClinux-dist/linux-2.4.x/drivers/net中





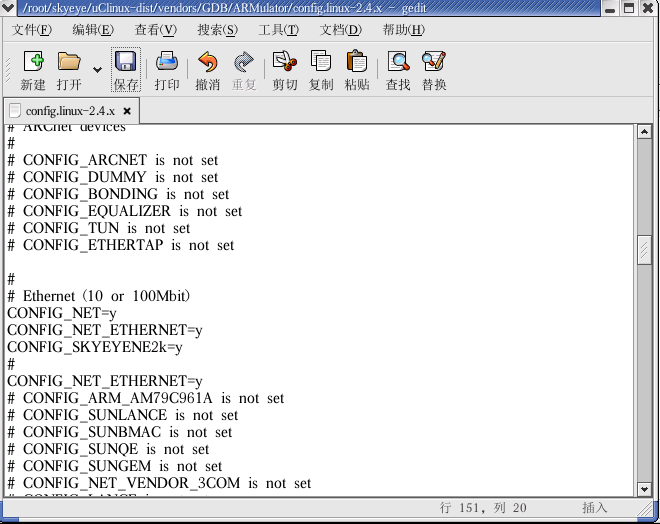
（2）修改config文件

/root/skyeye/uClinux-dist/vendors/GDB/ARMulator/config.Linux-2.4.x文件中添加：

CONFIG\_NET=y

CONFIG\_NET\_ETHERNET=y

CONFIG\_SKYEYENE2k=y

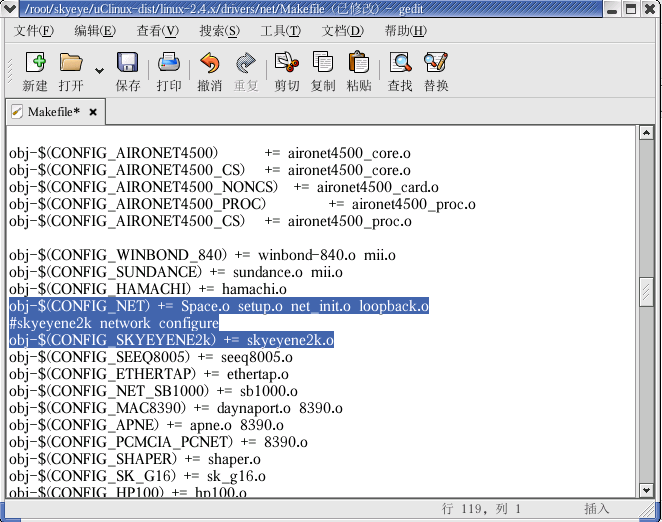


（3）修改Makefile文件

/root/skyeye/uClinux-dist/linux-2.4.x/drivers/net/Makefile文件添加：

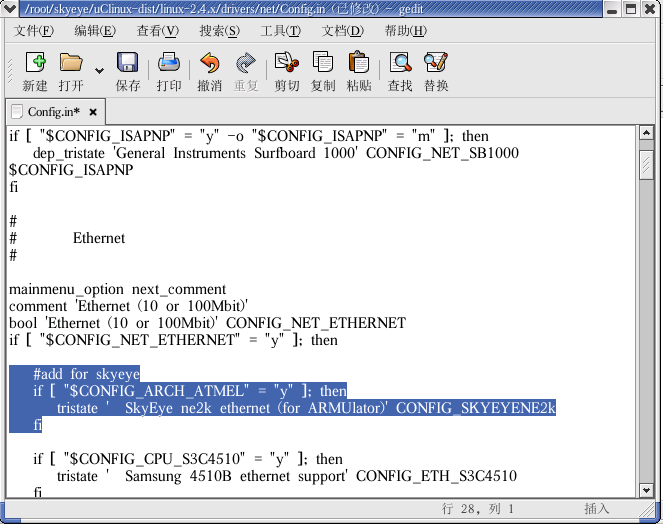
obj-$(CONFIG\_NET) += Space.o setup.o net\_init.o loopback.o

obj-$(CONFIG\_SKYEYENE2k) += skyeyene2k.o

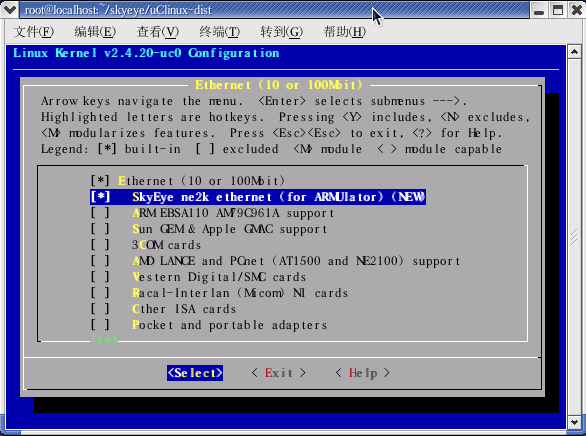


（4）修改Config.in文件

uClinux-dist/linux-2.4.x/drivers/net/Config.in文件添加：

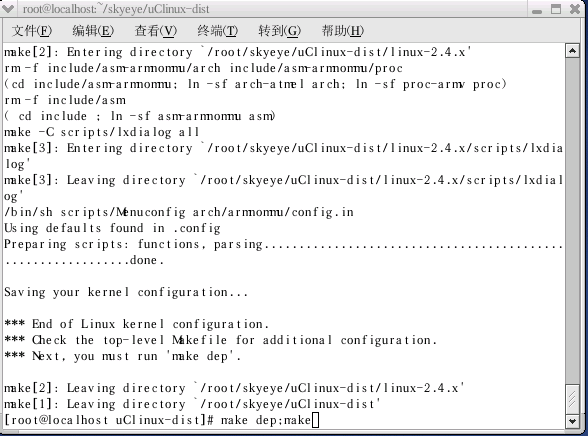


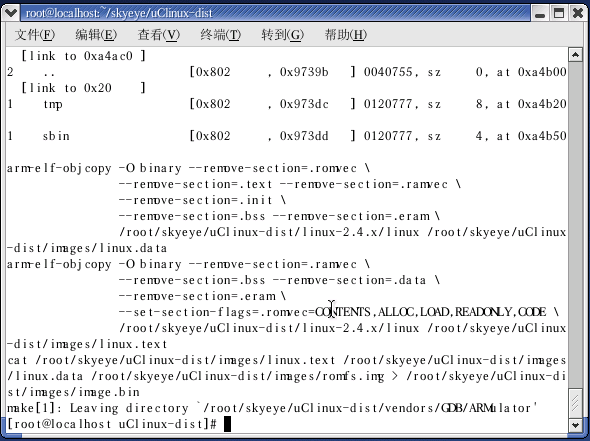
完成上述修改后，make menuconfig，继续之前的配置，选择Network device support -> Ethernet(10 or 100Mbit) -> SkyEye ethernet support(for ARMUlator)



#### 3.4.7 Exit退出保存

#### 3.4.8 编译生成：make dep;make

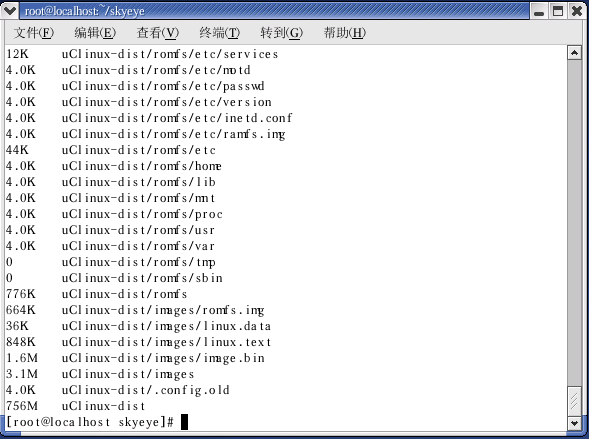




编译成功！

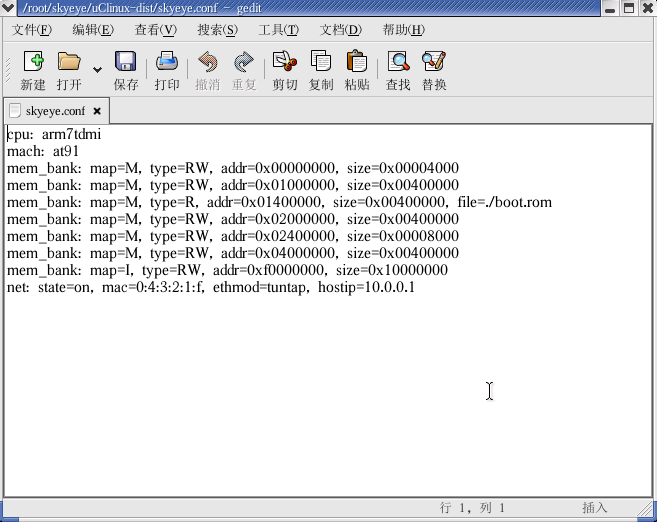
#### 3.4.9 查看代码量：du -ah uClinux-dist



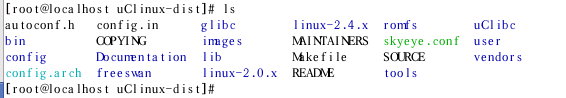


### 3.5 配置

#### 3.5.1 在工作目录(/root/skyeye/uClinux-dist)下建立专门用于基于AT91X40开发板的SkyEye硬件配置文件skyeye.conf:



ls查看：





#### 3.5.2建立文件系统的链接

在目录/root/skyeye/uClinux-dist目录下执行如下命令：

ln -s images/romfs.img boot.rom



### 3.6 运行

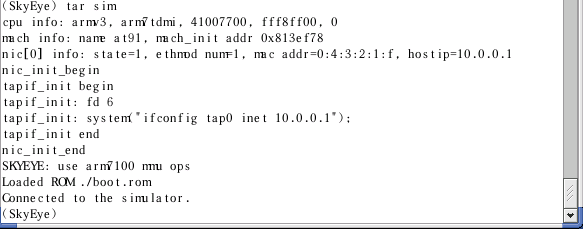
#### 3.6.1 进入/root/skyeye/uClinux-dist目录下，执行：

/usr/local/bin/skyeye linux-2.4.x/linux

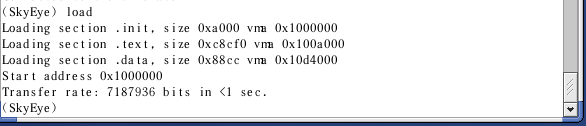
进入skyeye环境下

#### 3.6.2 在skyeye环境下

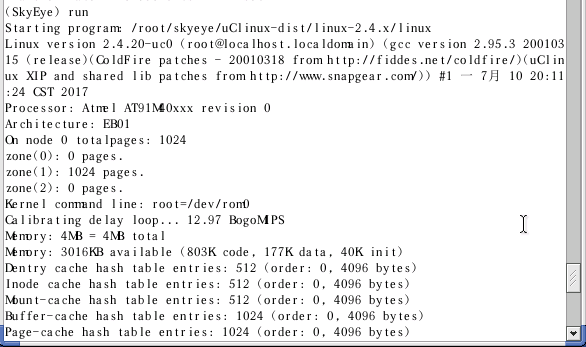
tar sim

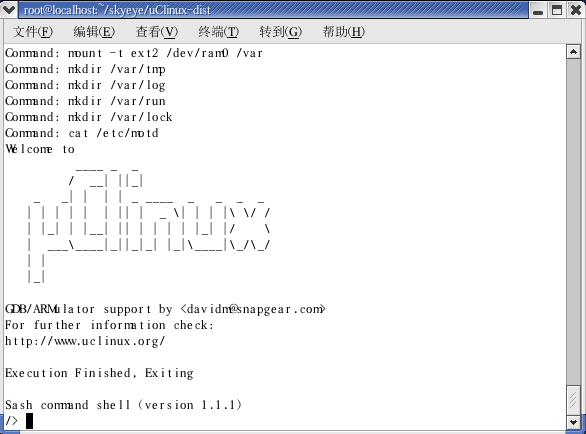


load



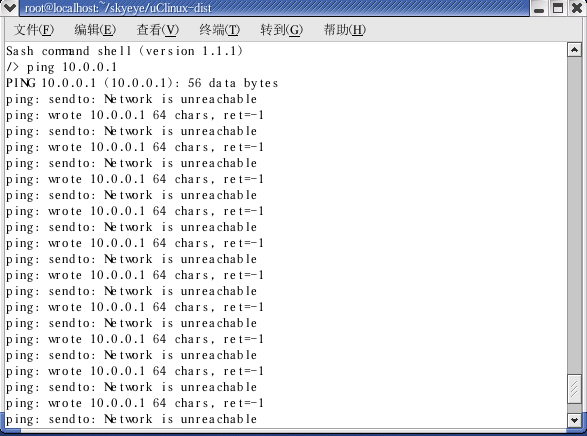
run





### 3.7 测试

#### 3.7.1 在skyeye内ping 10.0.0.1



发现ping不通

重新执行tar sim、load和run，修改ip后

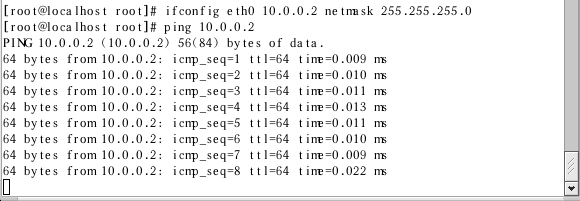


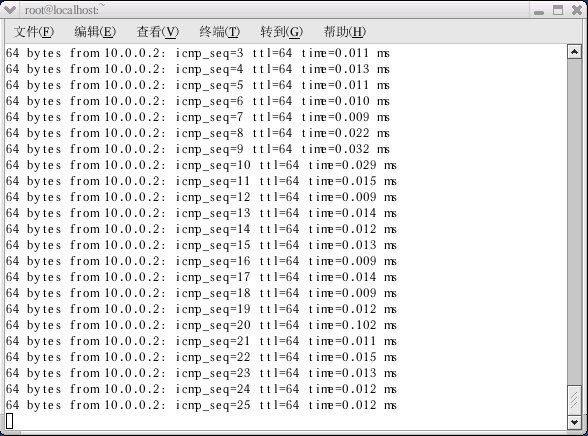
重新ping 10.0.0.1

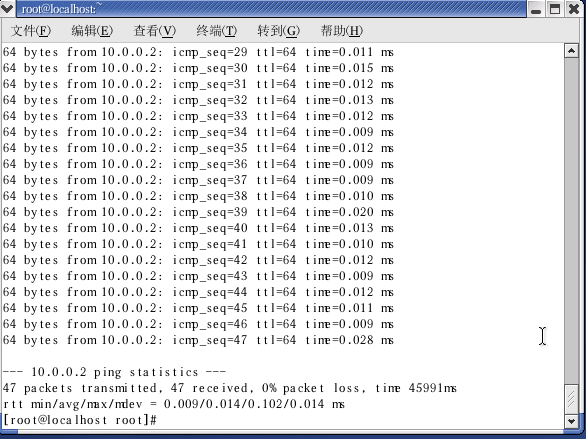


ping通！

#### 3.7.2 在skyeye外ping 10.0.0.2



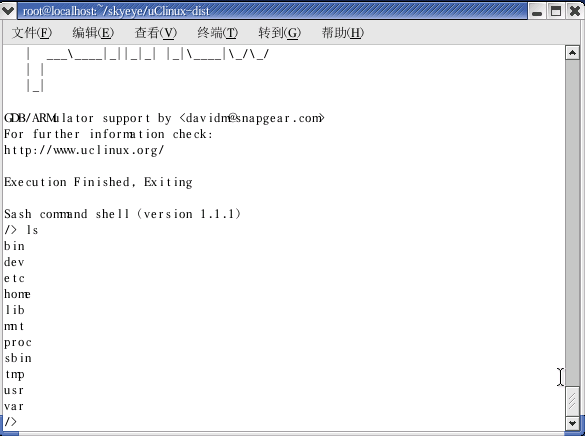




ping通，测试成功！

### 3.8 运行5条常用指令

ls：



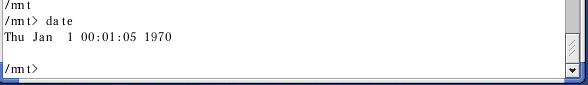
cd：



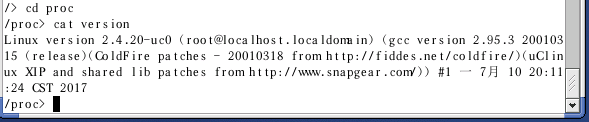
pwd：



date：



cat：



## 四、实验总结

此次实验耗时很长，首先是在安装uclinux时，在menuconfig配置时找不到以太网的配置选项，很苦恼，在网上历经千辛万苦后下载了其他文件解决了问题，然后在ping测试时一开始也没有ping通，通过ipconfig配置后解决了问题。

此次实验锻炼了我的动手实践能力，以及发现问题解决问题的能力，受益匪浅！

# 实验四 ARM/ucLinux-SkyEye下BusyBox集成——Shell小型化

## 一、实验目的

理解Linux根文件系统组织结构，将应用程序BusyBox放入Linux文件系统，在嵌入式ARM/ucLinux-SkyEye环境下，集成小型化shell BusyBox。

## 二、实验环境

虚拟机：VMware Workstation 12.1.1（Windows 10环境）

Linux操作系统：Red Hat Linux 9，Linux内核版本为2.4.20-8

硬件：Intel i586 PC机

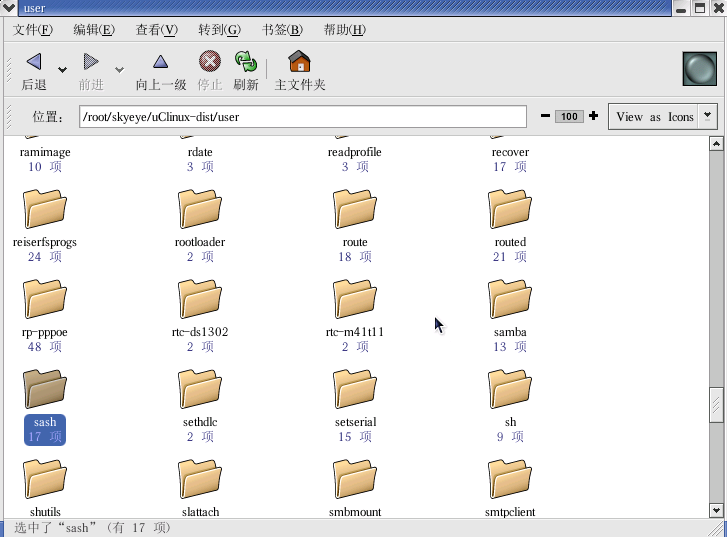
## 三、实验内容与步骤

参照按照参考文献[2]“6.3.3嵌入式应用软件BusyBox”、参考文献[1]“6.4.3小型化shell”的内容，用BusyBox替换原有的shell。

### 3.1 当前shell

当前ucliunx使用的shell为sach，sash对应的代码为uClinux-dist/user/sash



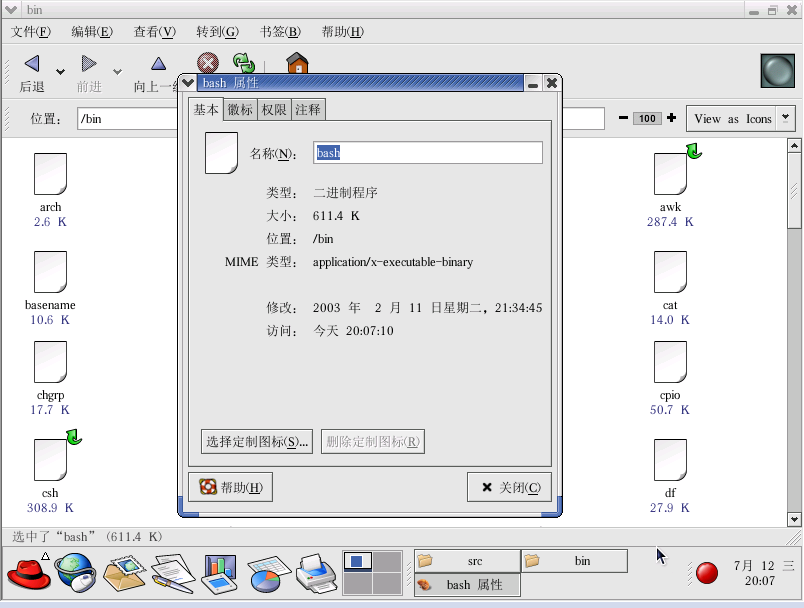




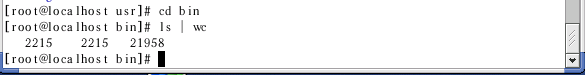
查看原本Linux使用的shell，命令为echo $0



查看bash的大小

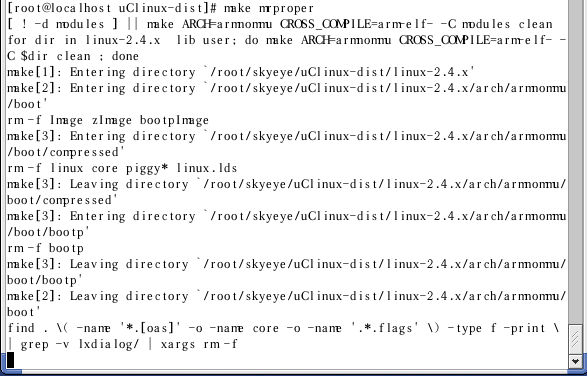


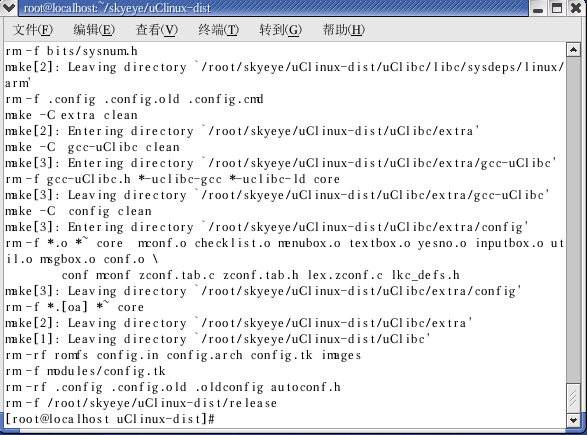
命令条数：



### 3.2 busybox替换shell

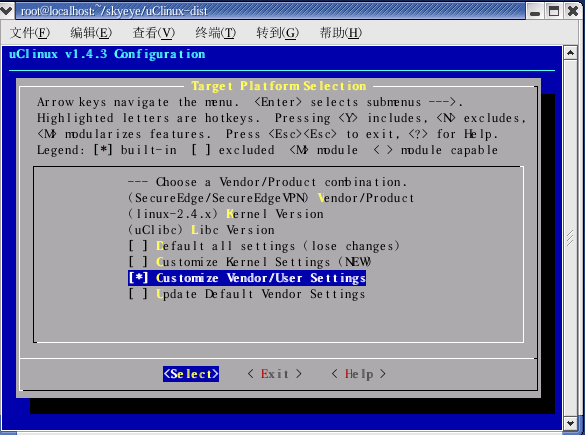
#### 3.2.1 内核归位：make mrproper



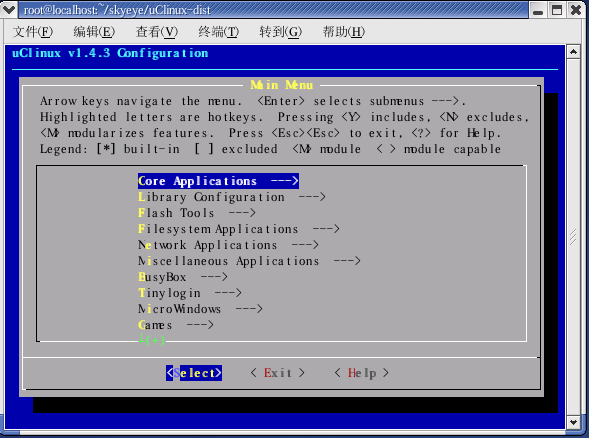


#### 3.2.2 make menuconfig设置uclinux内核，使其使用busybox

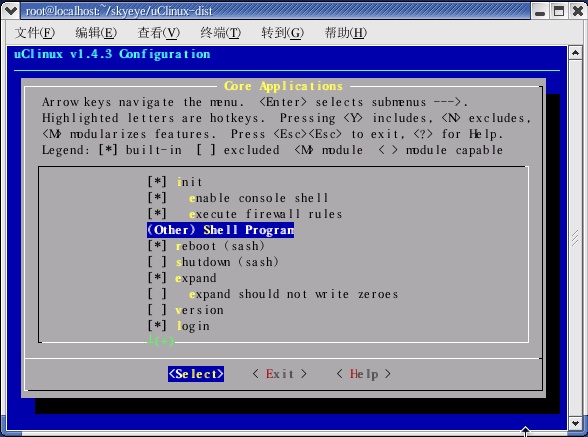
选择Customize Vendor/User Settings



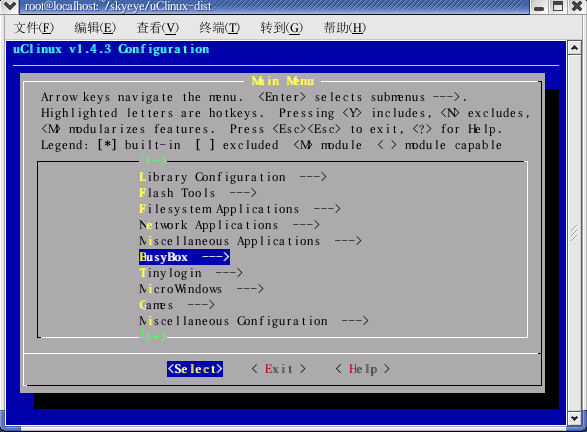
选择Core Applications



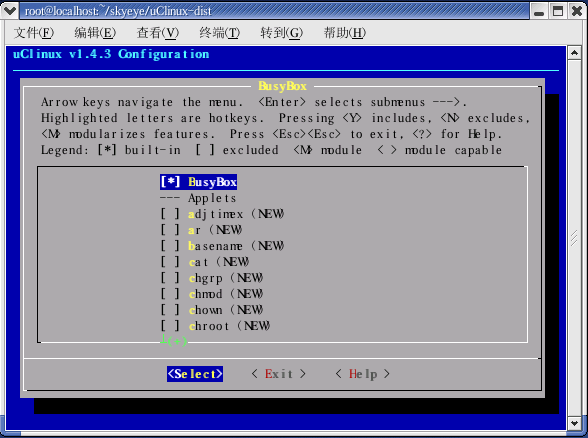
选择（Other）Shell Program

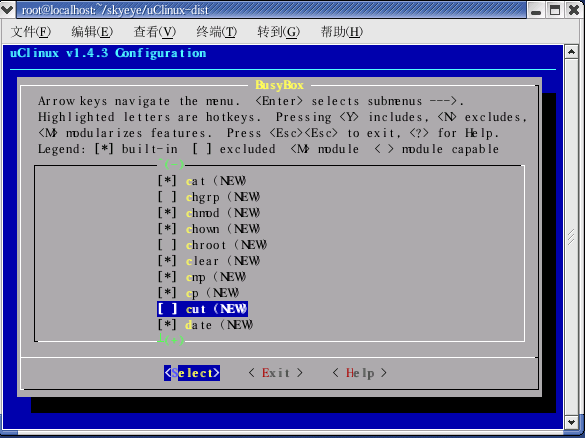


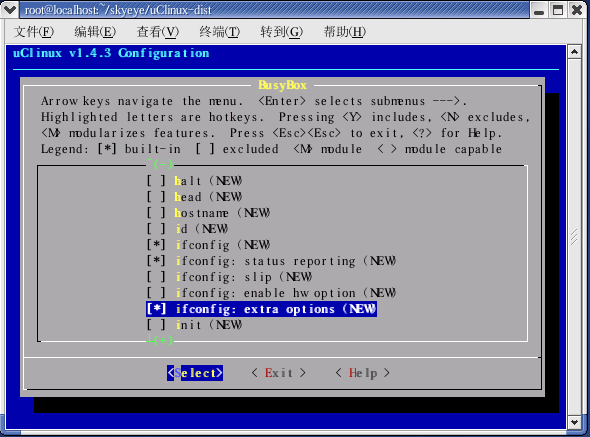
返回上一级，配置busybox

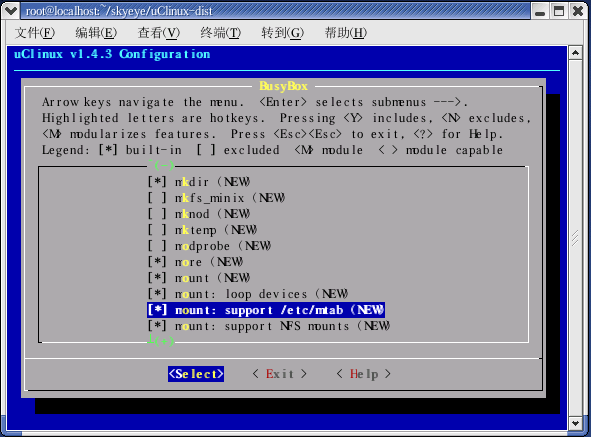


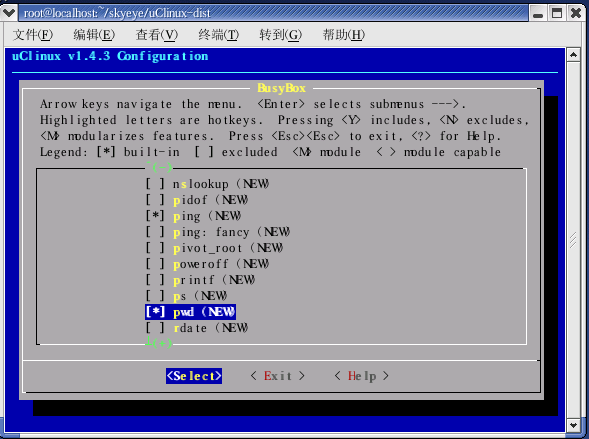
选择一些常见的指令：



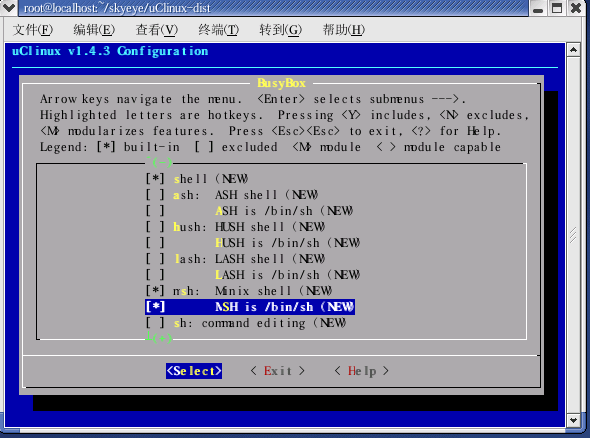




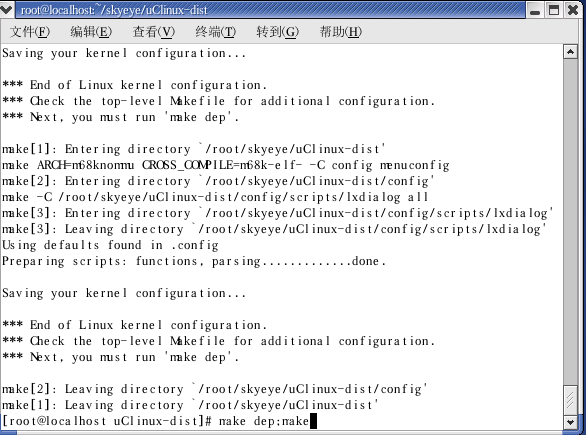




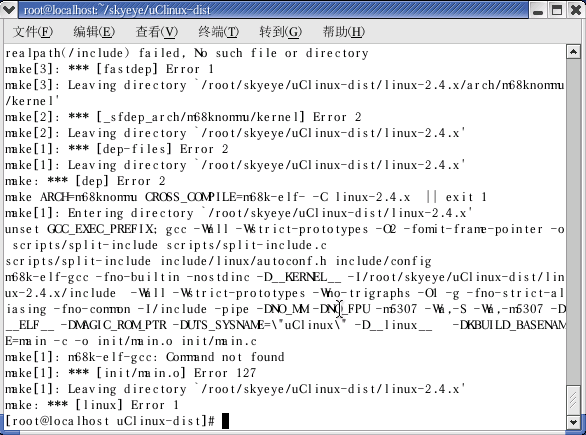
选择要使用的shell：



选择Exit，退出保存



#### 3.2.3执行 make dep;make

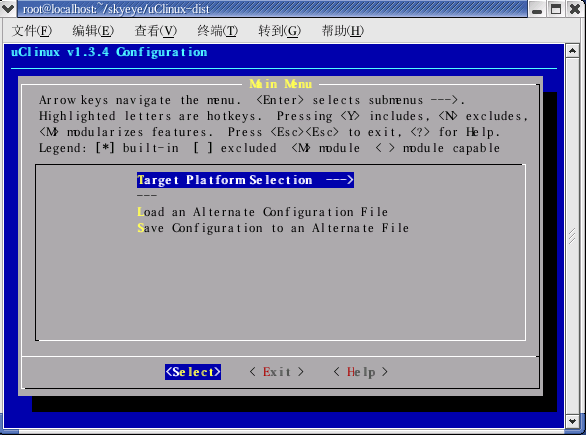


发现错误！

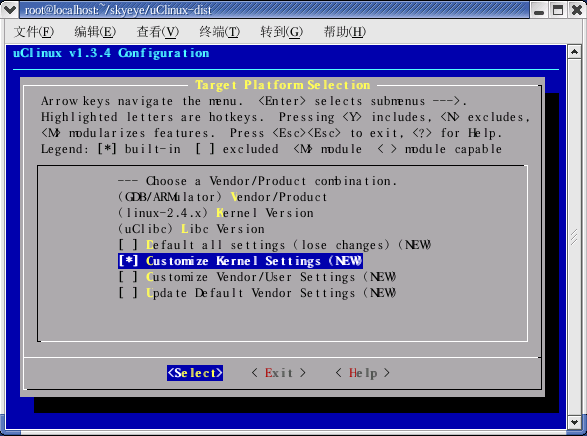
发现在Vendors/Product的配置还是要选择GDB/ARMulator

重新make menuconfig

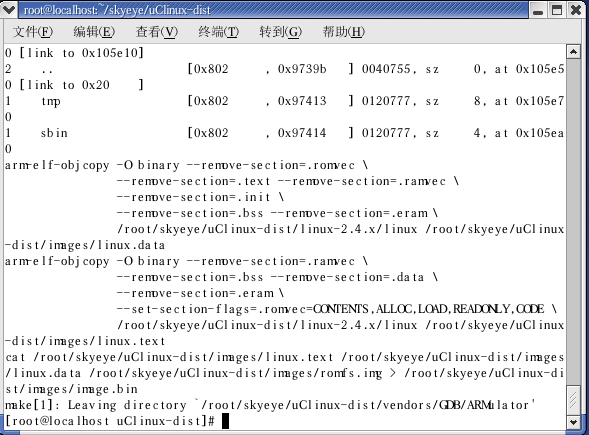




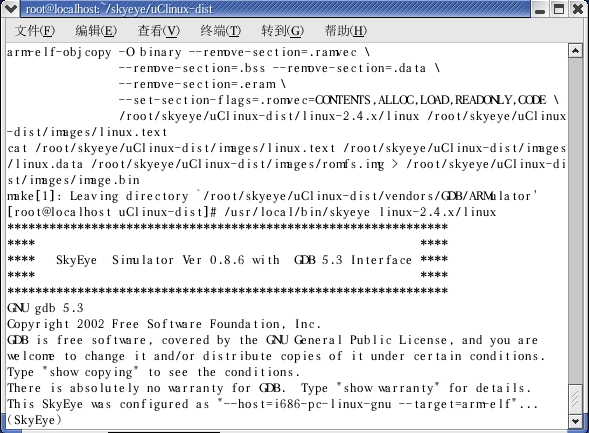


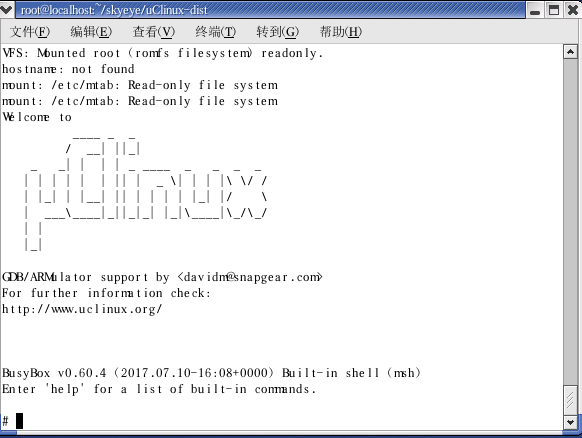


再次make;dep make，成功！

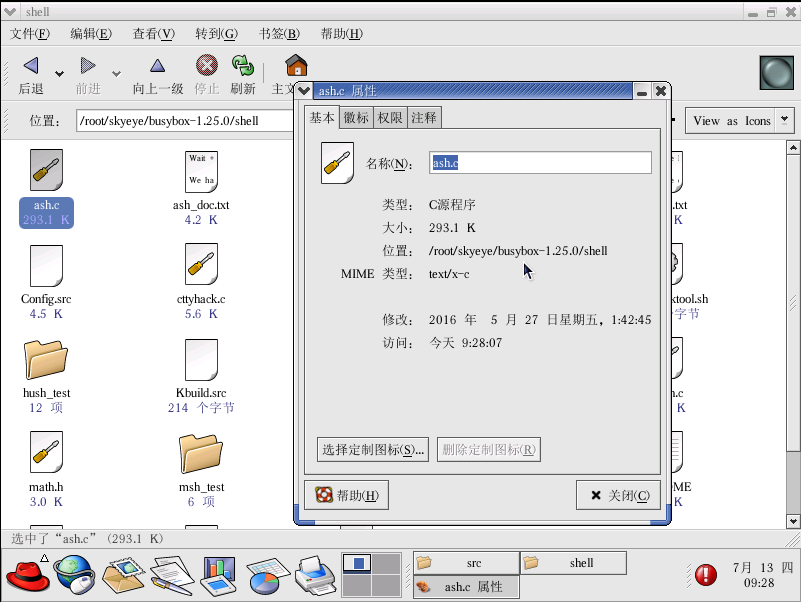


#### 3.2.4 运行skyeye，进入ucLinux，替换结果如下：

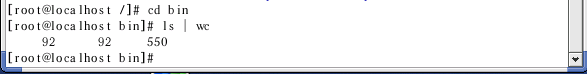




进入/root/skyeye/busybox-1.25.0查看shell大小：



命令条数：



### 3.3 代码测试

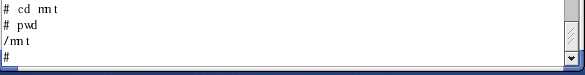
date：



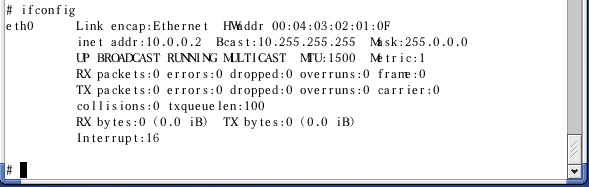
pwd：



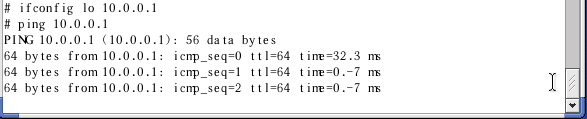
cd：



ifconfig：



ping：



命令可用！替换成功！

默认配置, 与uClibc静态链接后的busybox大小为: 1.1M；

剔除了不需要的内容, 与glibc动态链接后的busybox大小为: 382K

剔除了不需要的内容, 与uClibc静态链接后的busybox大小为: 509K

一共100多条指令

## 四、实验总结

深入了解了linux小型化，了解了busybox在uclinux中的替换配置方法，深感配置中必须要细致认真，出现一个小错误都可能影响全局结果。

# 实验五 Linux环境下Ramdisk技术及其实验

## 一、实验目的

ramdisk技术利用内存模拟硬盘空间，将一个内存块作为1个盘分区使用，可以提高访问速度。多某些特定的访问频率高的文件，将其存储于ramdisk上，可以提高访问性能。

在嵌入式系统中，利用ramdisk技术，可以实现应用程序预加载，即初始化完成后立即加载，而不是等到需要使用时才从硬盘读取，从而提高系统速度：将一些系统启动后经常使用的程序，如浏览器、Xwindows，利用ramdisk将整个应用程序的可执行文件及其库复制到内存，然后修改可执行程序加载的搜索路径，用ramdisk所在的目录代替原来的可执行文件的目录路径。

本实验通过Linux环境下ramdisk的配置使用，了解ramdisk的原理和实现机制，验证其对改善系统运行速度的有效性。

## 二、实验环境

硬件：Intel PC机

软件：Red Hat Linux 9

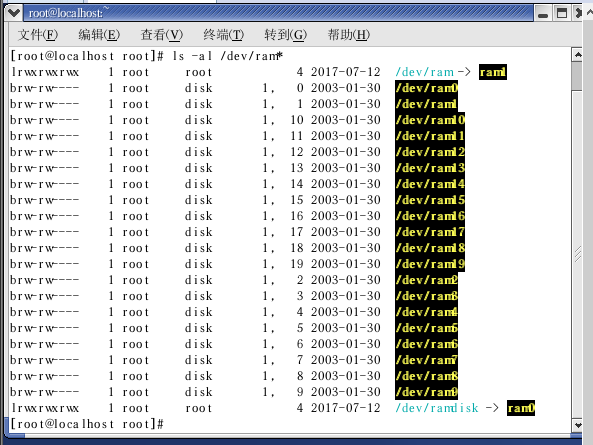
## 三、实验内容与步骤

按照参考文献[1]“6.3 ramdisk技术”相关内容，

### 3.1察看内核选项CONFIG\_BLK\_DEV\_RAM，确认Linux内核支持ramdisk选项

从Red Hat 6.0开始，默认安装本身就有对ramdisk的支持。因此，ramdisk使用非常简单，所要做的就是格式化一个ramdisk设备，然后安装（mount）到相应目录结构下即可。通过命令ls -al /dev/ram可以查看系统可利用的ramdisk设备的数目。在设置以后，这些ramdisk才发挥作用。

查看可用ramdisk设备：



### 3.2创建/格式化1个ramdisk 设备，并安装到文件系统的相应目录结构下

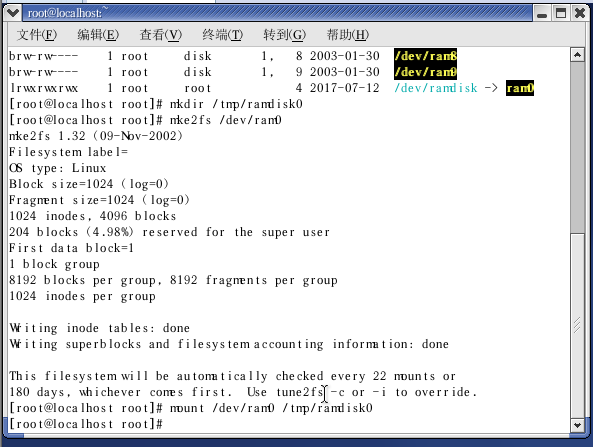
创建一个ramdisk的过程为

(1) 为ramdisk创建一个目录/挂载点，如mkdir /tmp/ramdisk0

(2) 格式化/创建一个文件系统, 如mke2fs /dev/ram0

(3) 将这个ramdisk安装(mount)到/tmp/ramdisk0目录下

然后就可以将该目录作为一个分区使用。



### 3.3运行命令“df –k /dev/ram0”，查看可使用的ramdisk空间大小

默认的ramdisk的大小为4MB=4096K bytes。在创建ramdisk文件系统时可以得到ramdisk大小的信息.

安装完ramdisk分区之后, 运行命令 df –k /dev/ram0 查看可以真正使用的ramdisk空间大小, 因为创建文件系统时会占用一些空间.

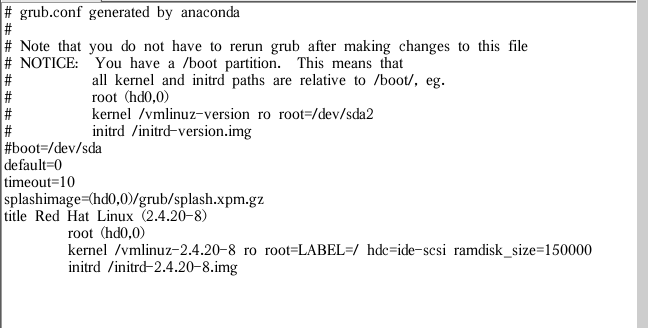


### 3.4 重新修改ramdisk空间大小

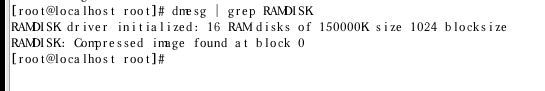
原有大小：



修改/etc/grub.conf



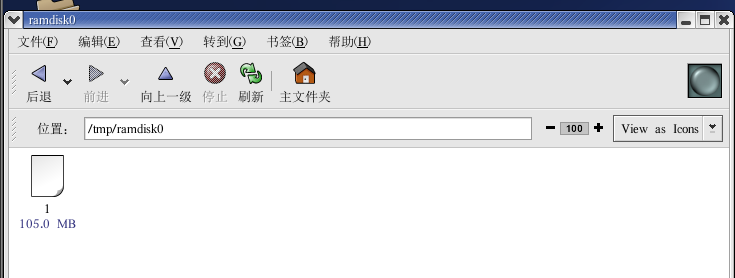
重启之后再次查看：



创一个大文件，使用指令**dd if=/dev/zero of=1 bs=1M count=105**创一个105M，名字为1，大小为105M的大文件。



把这个文件放到/tmp/ramdisk0中

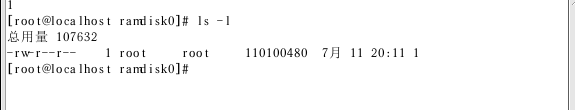


### 3.5 文件移动

**将Linux系统内常用的目录，这里选择/tmp/ramdisk0目录安装为ramdisk，通过打开一个大文件，并对文件进行读、写操作，观察验证系统运行速度是否加快。可以利用系统自身的时间测量函数。(截图为读取操作 写操作见源代码)**

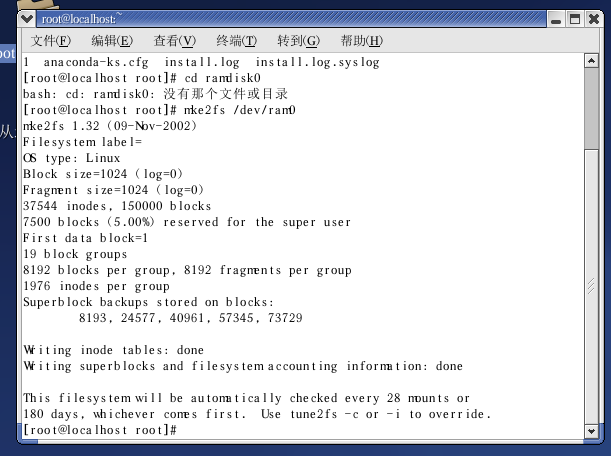
要求: 文件>100M（已经创建）

测试文件大小：

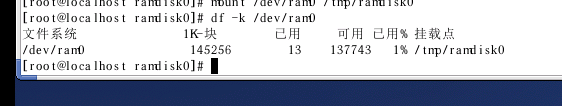


给ramdisk0分配ramdisk

将ramdisk0挂载ramdisk



运行命令“df –k /dev/ram0”查看是否挂载成功：



挂载成功

使用vi指令编写读文件的脚本程序，命名为test1.sh：



：wq保存。

把刚刚创建的120M大文件放进ramdisk0中



执行刚刚创的.sh文件，执行时间如下:





执行时间为27秒。

执行umount /dev/ram0 解挂。然后将大文件再次放进去，重复执行：





时间为27秒

**时间对比为：挂载比不挂载快 !**

作为ramdisk的1个应用，可以将/temp目录安装为ramdisk, 若系统中有很多程序使用/temp目录的话，则可以很明显地加快系统速度, 但是每次系统重新启动时, 这些暂时数据都会丢失。

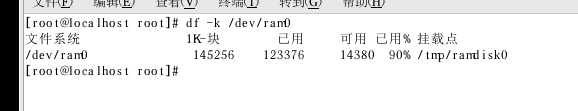
写入ramdisk测试比较：

文件大小>100M

写入文件的脚本程序，命名为test.sh



挂载：



挂载之后执行：



执行时间为：24秒

解挂后执行：



执行时间为：27秒

**时间对比为：挂载比不挂载快 !**

### 3.6 编写一个文件访问程序，对比将此文件分别存储于外设硬盘、ramdisk分区时，程序的执行速度。

要求：程序反复多次读、写文件，以便对比不同方式下的访问时间差异。

保存为test2.sh

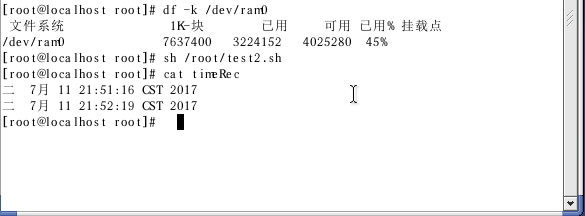


挂载之后执行：



时间：1分02秒

取消挂载后：



时间为：1分03秒

**时间对比为：挂载比不挂载快 !**

## 四、实验总结

这次实验进行了对挂载和不挂载状态下对文件读写操作，并且进行对比。挂载后使得运行时间大大缩短。让我学会一个新的方法运行程序。让我有了很大的感触。

# 实验六 Linux环境下嵌入式数据库mSQL的集成

## 一、实验目的

实时嵌入式应用常常需要数据库的支持。虽然很多情况下可以用文件方式实现部分数据库功能，但是当应用程序需要执行一些比较复杂的数据操作时，文件方式就无能为力了，更为合适的方式是采用适合实时嵌入式硬件和操作系统平台的嵌入式数据库系统。

要求学生通过本实验了解嵌入式数据库的基本特点。在Linux环境下安装配置嵌入式数据库mSQL，建立简单的数据库，利用SQL语言和mSQL API实现简单的数据库访问功能。

## 二、实验环境

虚拟机：VMware Workstation 12.1.1（Windows 10环境）

Linux操作系统：Red Hat Linux 9，Linux内核版本为2.4.20-8

硬件：Intel i586 PC机

## 三、实验内容与步骤

按照参考文献[3]“第14章嵌入式数据库”相关内容，

### 3.1 下载安装包，在Linux环境下正确安装

从mSQL网站www.hughes.com.au下载一份以源代码方式发布的mSQL软件包，

以源代码方式发布的mSQL的安装步骤如下：

1、在desktop解压压缩包：

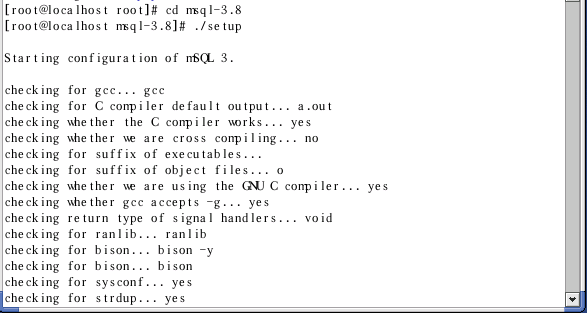
tar -zvxf msql-3.8.tar.gz

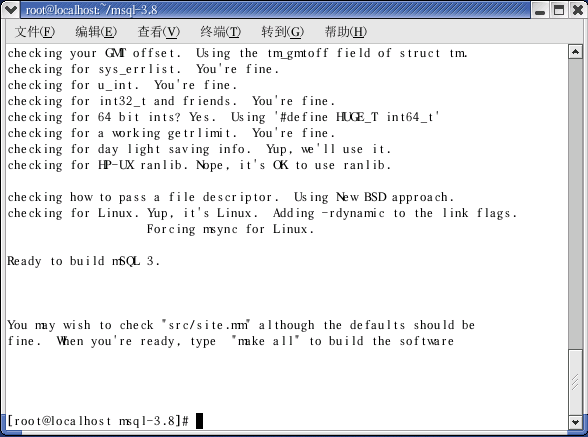
该命令会在当日安目录下建立一个名为msql-3.8的文件夹，它用于存放所有的发布文件，包括源代码目录src和文档目录doc。接下来的步骤要使用setup程序来设置后面的编译选项。

2、./setup：

该命令会将一些编译选项保存在src/site.mm文件中，如果用户需要改变mSQL程序的安装路径以及C编译器的类型，可以修改该文件中的对应内容。程序的默认安装路径为/usr/local/msql3.

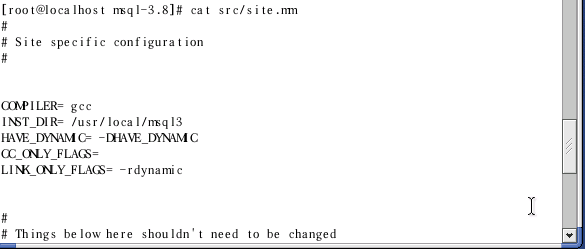
进入目录cd msql-3.8，执行./setup

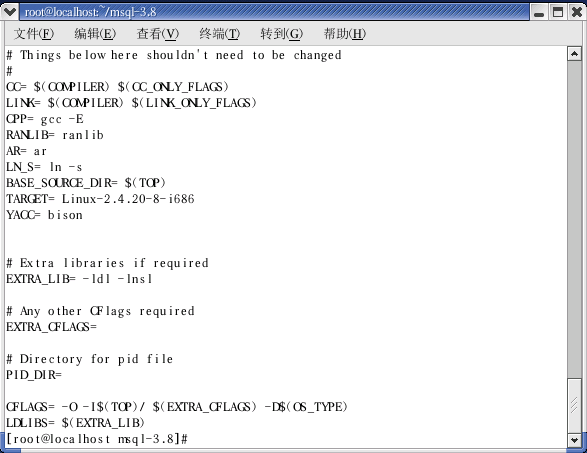




查看site.mm文件：

cat src/site.mm

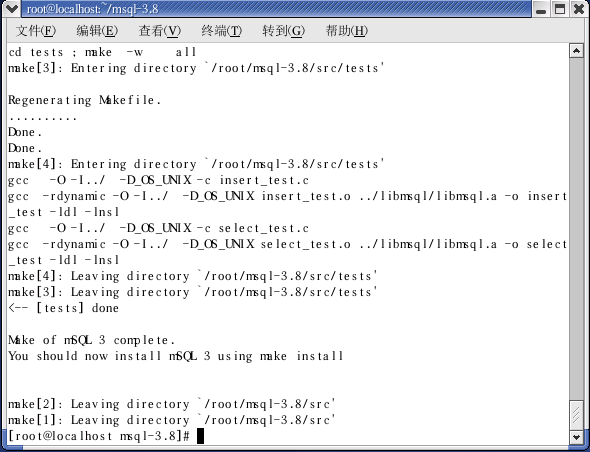




编译mSQL的源程序，命令如下：

make all

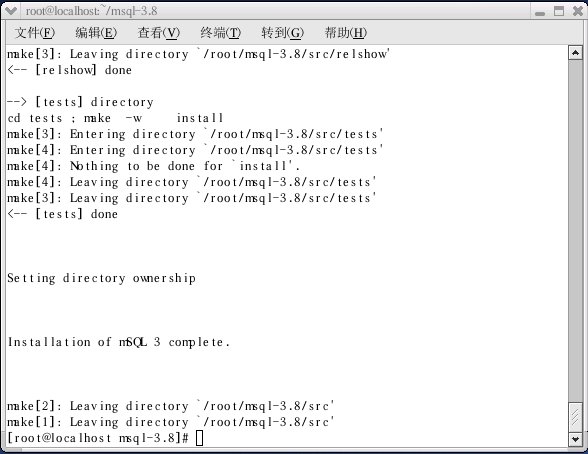




编译完成而且正确，则可以安装mSQL，命令如下：

make install





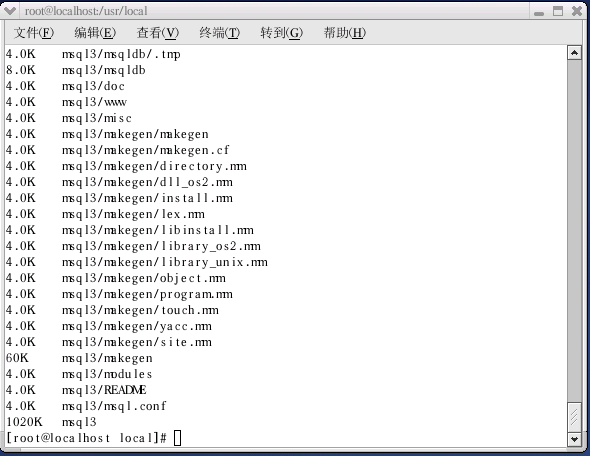
mSQL正确被安装在系统中

查看代码量大小，命令如下：

cd /usr/local

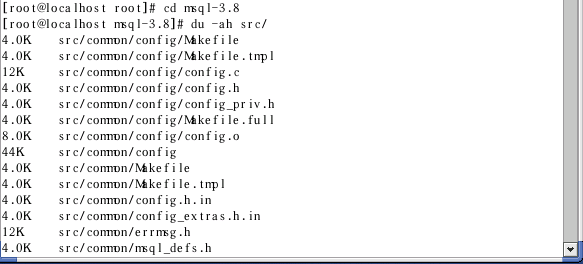
du -ah msql3

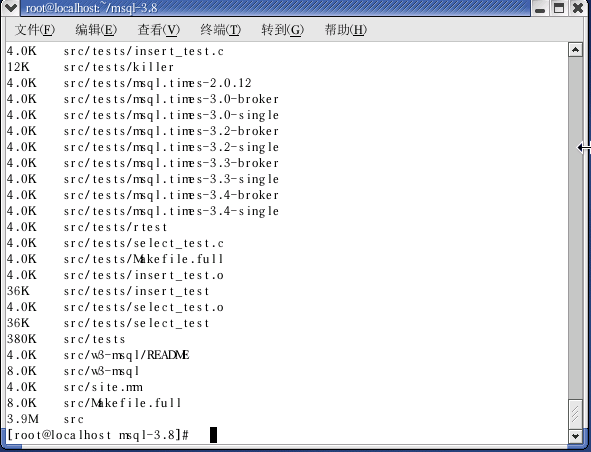




cd /msql-3.8

du -ah src/



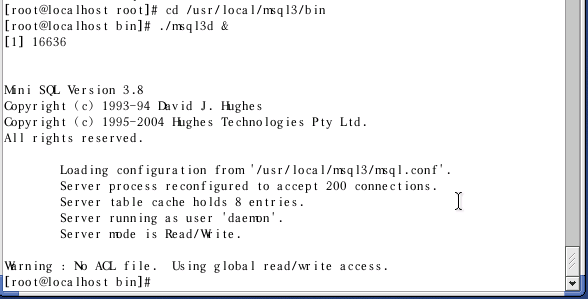


### 3.2 建立数据库

用mSQL自带工具检查软件安装，并面向具体领域，建立数据库, 如GSM/GPRS移动通信网络配置数据库

在使用msql之前需要进行操作：

进入/usr/local/msql3/bin目录，执行：./msql3d &

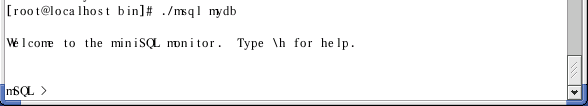


这样就以后台方式启动了mSQL服务器系统，从而开始进行具体的数据库创建、查询等操作。

创建数据库mydb：

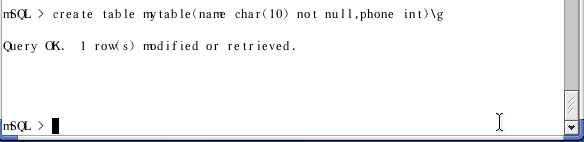


进入数据库mydb：



创建table：

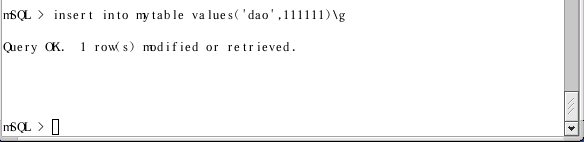
create table mytable(name char(10) not null,phone int)\g



### 3.3 利用SQL语句访问数据库系统，验证所安装系统的正确性

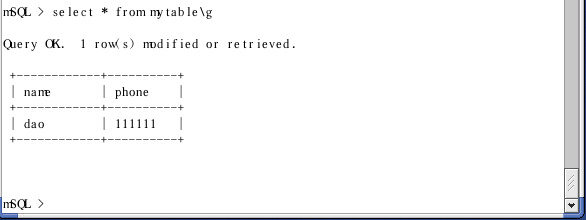
插入数据：

insert into mytable values('dao',111111)\g



查询数据：

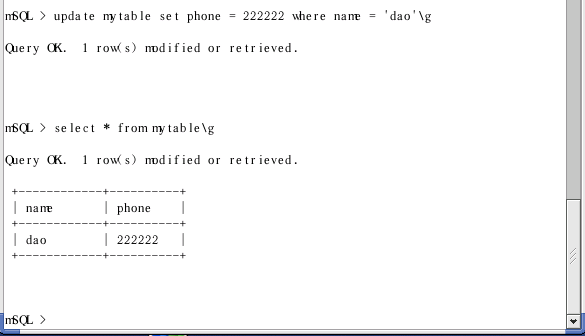
select \* from mytable\g



修改数据：

update mytable set phone = 123456 where name = 'dao'\g

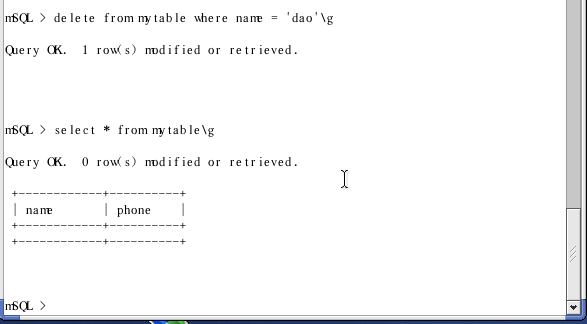
select \* from mytable\g



删除数据：

delete from mytable where name = 'dao'\g

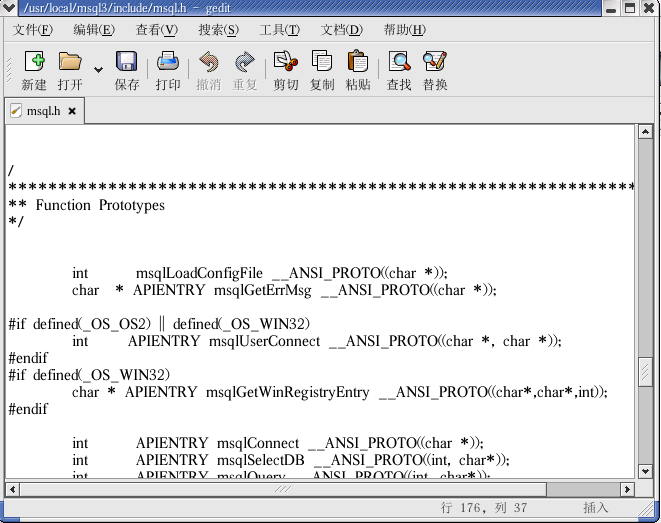
select \* from mytable\g

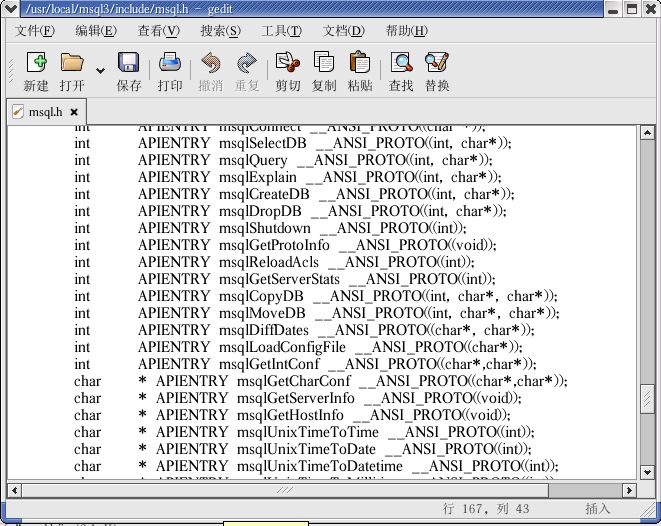


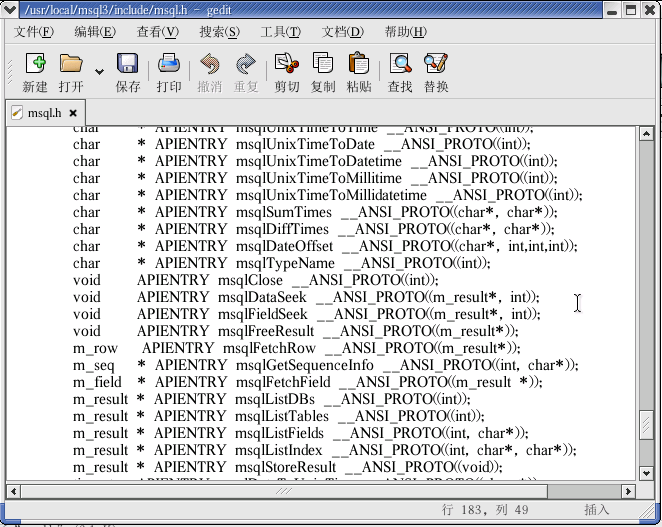
### 3.4 参考示范程序，利用mSQL API，编写实现数据库访问功能的C语言程序。

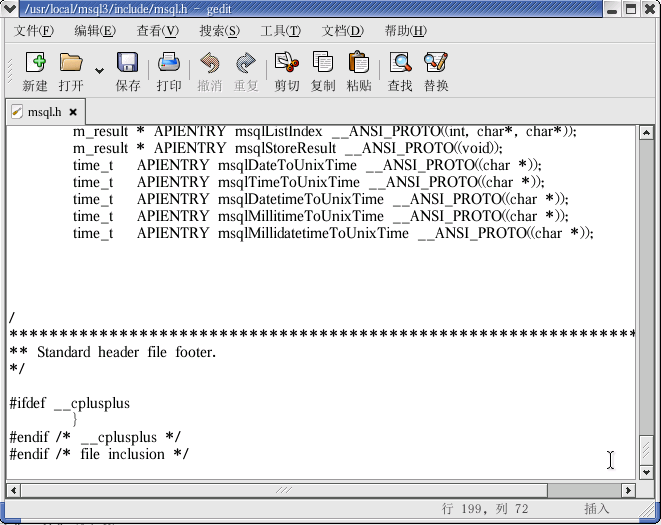
mSQL的API函数库名称为libsql.a，一般位于mSQL安装路径下的lib目录中，库中的函数在msql.h中定义。用户在编写程序时，应包含该头文件，该文件一般位于mSQL安装路径下的inclube目录，如/usr/local/msql3/include中。另外，在对C程序进行编译链接的时候，应加上链接参数。

头文件API：

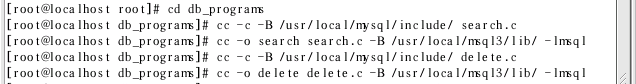


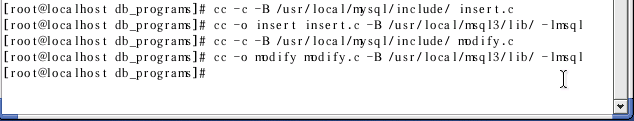






准备编译程序：



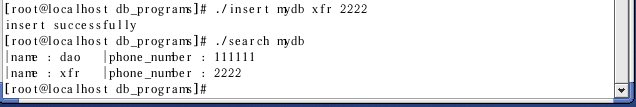


编译程序

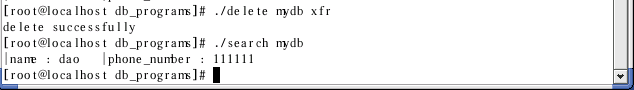
查询：



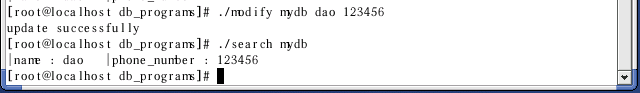
插入：



删除：



更新：



## 四、实验总结

此实验通过在Red Hat下安装使用mSQL来实现对数据库的操作，嵌入式系统中安装mSQL占用内存小，支持SQL查询语句以及SQL标准子集，并且提供了简明的API接口，可以在高级语言中进行调用。在实验中，我利用mSQL的API编写了C语言程序 来访问数据库进行增删改查的操作，我对mSQL的功能和配置有了更加深入的理解，同时，对于嵌入式系统下使用的嵌入式数据库也有了一定的认识。通过这次实验，我能够在Linux系统下进行数据库的各项操作以及调用，由此体会到嵌入式的意义以及嵌入式系统的含义。在实验中我也遇到了一些问题，如实际编写c程序时需要对参考程序的参数进行修改；C程序中需要将%d改为%s输出为字符串的形式，否则数据不对。发现了这些问题并解决后，我的动手实践能力提高了，解决问题的能力也提高了！

## 五、源代码

**search.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include "/usr/local/msql3/include/msql.h"

#define VERBOSE

/\*SQL query \*/

#define SELECT\_QUERY "select \* from mytable"

int main(argc,argv)

int argc;

char \*argv[];

{

int sock, numrow, num;

char qbuf [160],

\* host,//host name

\* db;//database name

m\_result \*res;

m\_row row;

/\*get execute parameters i.e. host & db\*/

if (argc == 5)

{

host = argv[2];

db = argv[3];

}

else

{

host = NULL;

db = argv[1];

}

/\*connect to msql server\*/

if ((sock = msqlConnect(host)) < 0)

{

printf("Could not connect\n");

perror("");

exit (1);

}

/\*select db\*/

if (msqlSelectDB (sock,db) < 0)

{

printf("Could not connect\n");

}

sprintf(qbuf,SELECT\_QUERY);

if (numrow = msqlQuery(sock,qbuf) < 0)

{

printf("query failed\n");

exit (1);

}

/\*get query result\*/

res = msqlStoreResult();

numrow = msqlNumRows(res);

/\*show result\*/

for (num = 0; num < numrow; num++)

{

msqlDataSeek(res,num);

row = msqlFetchRow(res);

printf("|name : %s |phone\_number : %s\n",row[0],row[1]);

}

/\*close the connection\*/

msqlClose (sock);

exit (0);

return 0;

}

**delete.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include "/usr/local/msql3/include/msql.h"

#define VERBOSE

/\*SQL query \*/

#define DELETE\_QUERY "delete from mytable where name="

int main(argc,argv)

int argc;

char \*argv[];

{

int sock, numrow, num;

char qbuf [160],

\* host,//host name

\* db;//database name

m\_result \*res;

char aaa[160];

strcpy(aaa,DELETE\_QUERY);

m\_row row;

/\*get execute parameters i.e. host & db\*/

if (argc == 3)

{

host = NULL;

db = argv[1];

strcat(aaa,"'");

strcat(aaa,argv[2]);

strcat(aaa,"'");

strcat(aaa,"\0");

}

else

{

host=NULL;

db=argv[1];

}

/\*connect to msql server\*/

if ((sock = msqlConnect(host)) < 0)

{

printf("Could not connect\n");

perror("");

exit (1);

}

/\*select db\*/

if (msqlSelectDB (sock,db) < 0)

{

printf("Could not connect\n");

}

//sprintf(qbuf,SELECT\_QUERY);

if (numrow = msqlQuery(sock,aaa) <= 0)

{

printf("delete failed\n");

exit (1);

}

else

{

printf("delete successfully\n");

}

/\*get query result\*/

/\*show result\*/

/\*close the connection\*/

msqlClose (sock);

exit (0);

return 0;

}

**modify.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include "/usr/local/msql3/include/msql.h"

#define VERBOSE

/\*SQL query \*/

#define UPDATE\_QUERY "update mytable set phone="

int main(argc,argv)

int argc;

char \*argv[];

{

int sock, numrow, num;

char qbuf [160],

\* host,//host name

\* db;//database name

m\_result \*res;

char aaa[160];

strcpy(aaa,UPDATE\_QUERY);

m\_row row;

/\*get execute parameters i.e. host & db\*/

if (argc == 4)

{

host = NULL;

db = argv[1];

strcat(aaa,argv[3]);

strcat(aaa," where name= '");

strcat(aaa,argv[2]);

strcat(aaa,"'");

}

else

{

host=NULL;

db=argv[1];

}

/\*connect to msql server\*/

if ((sock = msqlConnect(host)) < 0)

{

printf("Could not connect\n");

perror("");

exit (1);

}

/\*select db\*/

if (msqlSelectDB (sock,db) < 0)

{

printf("Could not connect\n");

}

//sprintf(qbuf,SELECT\_QUERY);

if (numrow = msqlQuery(sock,aaa) <= 0)

{

printf("update failed\n");

exit (1);

}

else

{

printf("update successfully\n");

}

/\*get query result\*/

/\*show result\*/

/\*close the connection\*/

msqlClose (sock);

exit (0);

return 0;

}

**insert.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include "/usr/local/msql3/include/msql.h"

#define VERBOSE

/\*SQL query \*/

#define INSERT\_QUERY "insert into mytable values"

int main(argc,argv)

int argc;

char \*argv[];

{

int sock, numrow, num;

char qbuf [160],

\* host,//host name

\* db;//database name

m\_result \*res;

char aaa[160];

strcpy(aaa,INSERT\_QUERY);

m\_row row;

/\*get execute parameters i.e. host & db\*/

if (argc == 4)

{

host = NULL;

db = argv[1];

strcat(aaa,"('");

strcat(aaa,argv[2]);

strcat(aaa,"',");

strcat(aaa,argv[3]);

strcat(aaa,")");

strcat(aaa,"\0");

}

else

{

host=NULL;

db=argv[1];

}

/\*connect to msql server\*/

if ((sock = msqlConnect(host)) < 0)

{

printf("Could not connect\n");

perror("");

exit (1);

}

/\*select db\*/

if (msqlSelectDB (sock,db) < 0)

{

printf("Could not connect\n");

}

//sprintf(qbuf,SELECT\_QUERY);

if (numrow = msqlQuery(sock,aaa) <= 0)

{

printf("insert failed\n");

exit (1);

}

else

{

printf("insert successfully\n");

}

/\*get query result\*/

/\*show result\*/

/\*close the connection\*/

msqlClose (sock);

exit (0);

return 0;

}