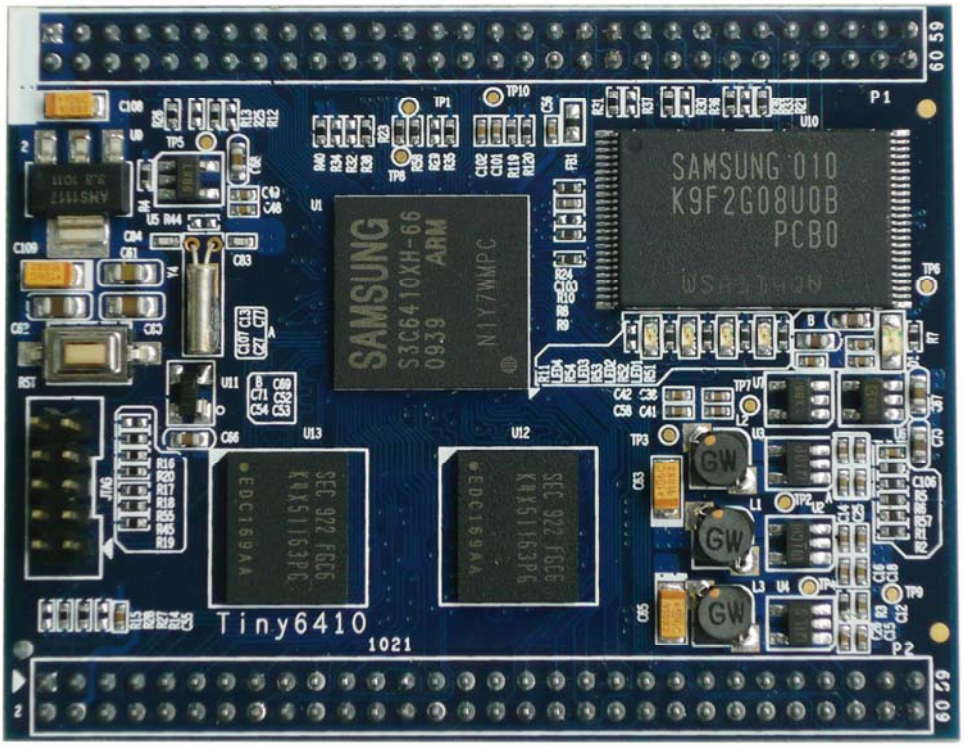
Tiny6410 Linux 开发指南

版本：2011-1-4

(本手册正在不断更新中，建议您到网站下载最新版本)



版 权 声 明

敬告：

在售开发板的手册会经常更新，请在 http://www.arm9.net网站查看最近更新，并下载最新手册，不再另行通知。

更新说明：

日期	说明
2011-1-4	修正了几处笔误，并重新对说明了 WiFi 无线网卡的使用。
2010-11-27	本手册第一次发布，任何问题请反馈至 capbily@163.com

目　录

Tiny6410 Linux 开发指南

Tiny6410 的软件和Mini6410 是完全兼容的，因此路径的设置，缺省的配置文件等，都沿用了Mini6410 的手册说明，可能会根据实际情况有稍微不同，若有疑问，请和我们联系：capbily@163.com

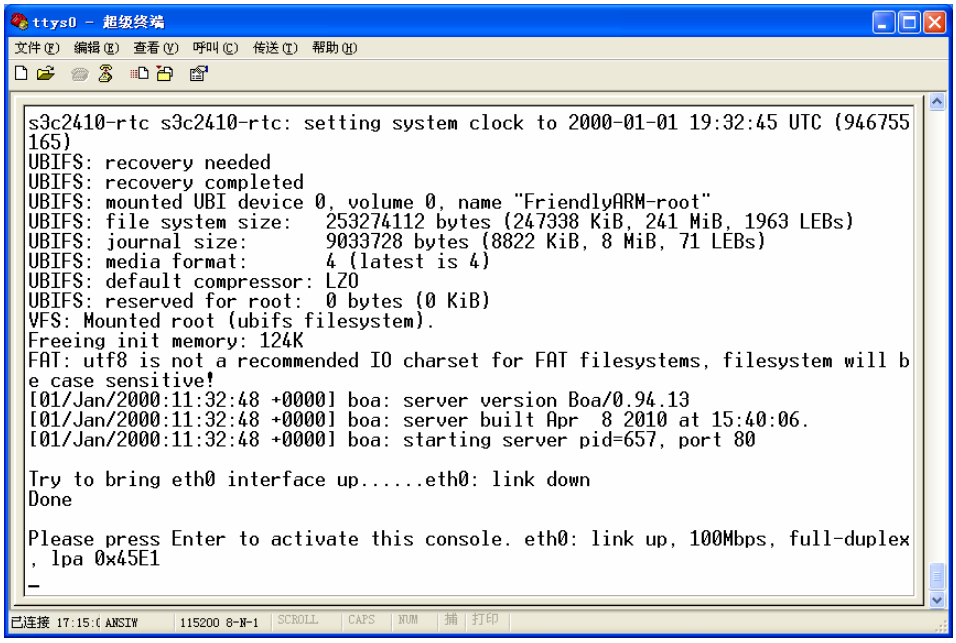
1.1 初试 Linux 之图形界面 Qtopia-2.2.0，Qtopia4 和 QtE-4.7.0 系统

请参考“Tiny6410 功能测试指南”

1.2 通过串口终端操作开发板

说明：每个期望学习 Linux 的嵌入式爱好者都应该学会熟练使用终端控制台的操作，所有平台的 Linux 指令都是相似的，超过 99%的命令是相同的。进行本小节的操作之前，请先按照“Tiny6410 刷机指南”的步骤正确设置好超级终端。

下图是通过串口终端显示的 Linux 登录界面，实际可能与此不完全相同，但基本都是类似的，根据提示，按下回车，就可以开始 Linux 控制台之旅了。

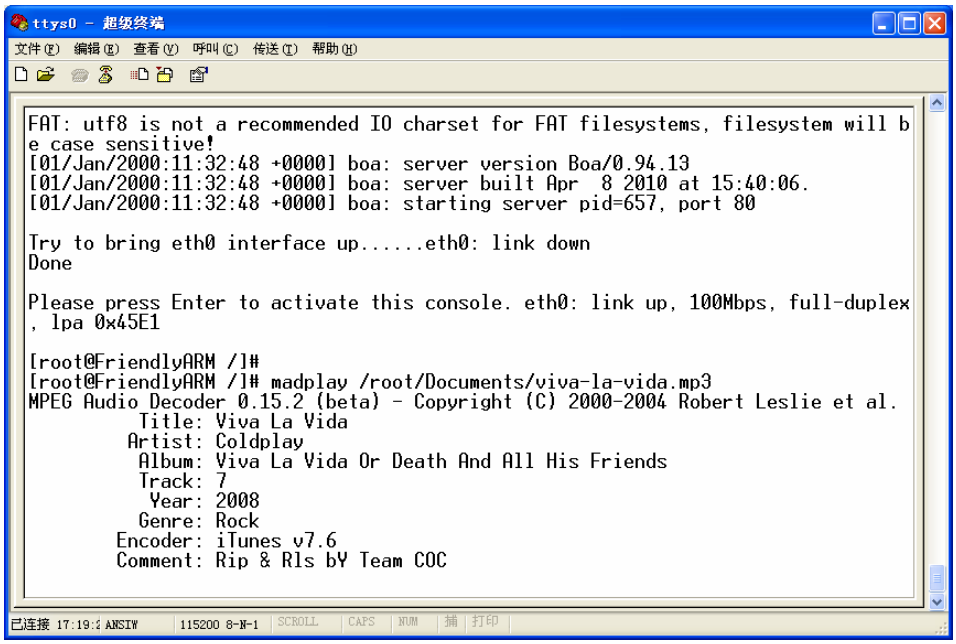


4.2.1 播放 mp3

madplay 是我们移植的一个基于控制台下的 mp3 播放器。它有多种播放控制模式，最简单的使用方法是：

#madplay your.mp3

该命令将以缺省模式播放 your.mp3 文件(开发板中并无 your.mp3 文件，这里只是举例说明)。可以运行“madplay -h”查看其使用帮助，下面是播放开发板中预装包含的一首 mp3 的截图：



需要说明的是：在 Linux-2.6.36 内核中，我们采用了 ALSA 接口的音频驱动，并且此处我们移植的 madplay 也采用了此接口实现播放，并且开发板中已经内置了 ALSA 库。

1.2.2 如何中止程序的运行

要中止程序的运行，可以在终端控制台下同时按下 **Ctrl+c**，注意：先按 **Ctrl**，不要放开，再按下 **c** 键即可。

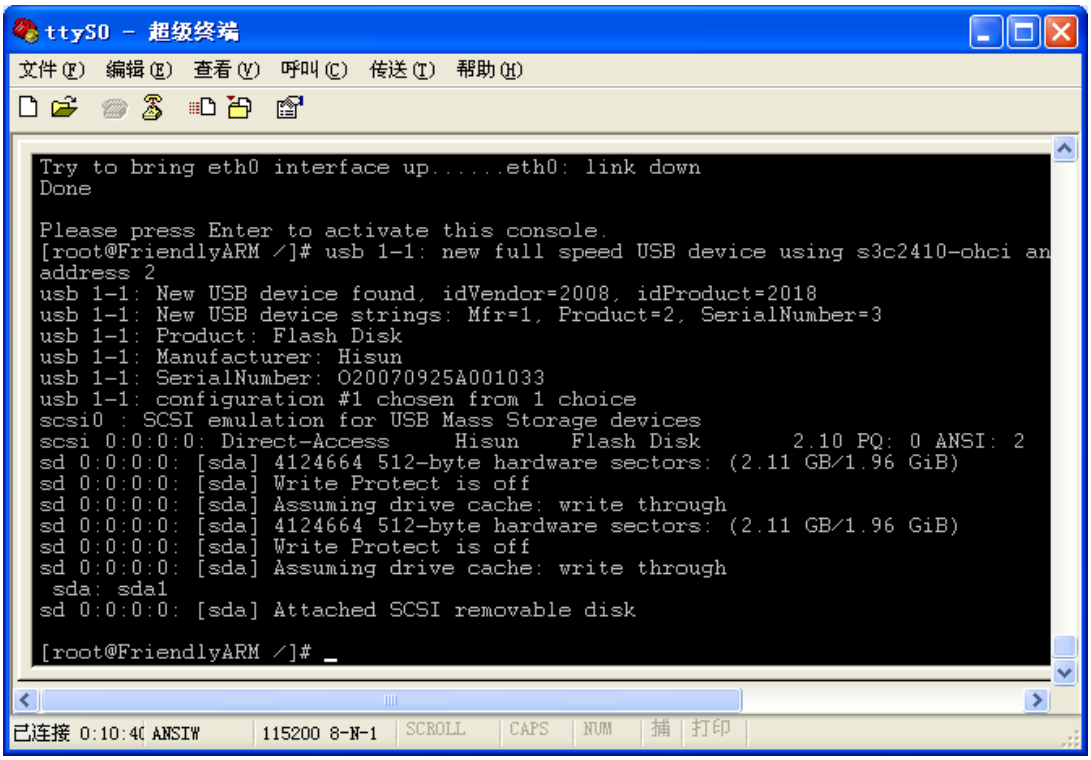
例如：我们刚刚使用 madplay 命令播放了 mp3，如果要中止这个程序的运行，可以按下 **Ctrl+c** 键。

另外，如果程序是在后台运行，可以使用 **kill** 命令杀掉该进程

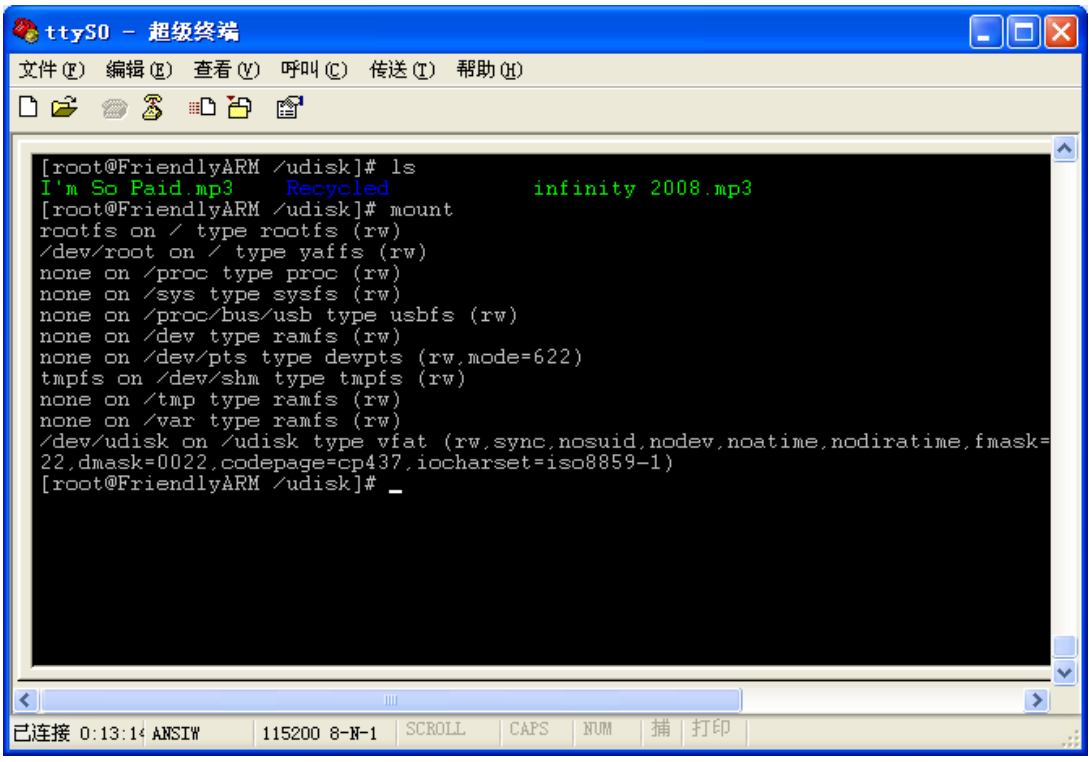
1.2.3 使用优盘/移动硬盘

插入优盘之后，系统会自动创建一个/udisk 目录，并自动挂载优盘到上面，此时在串

口会出现类似如下信息：

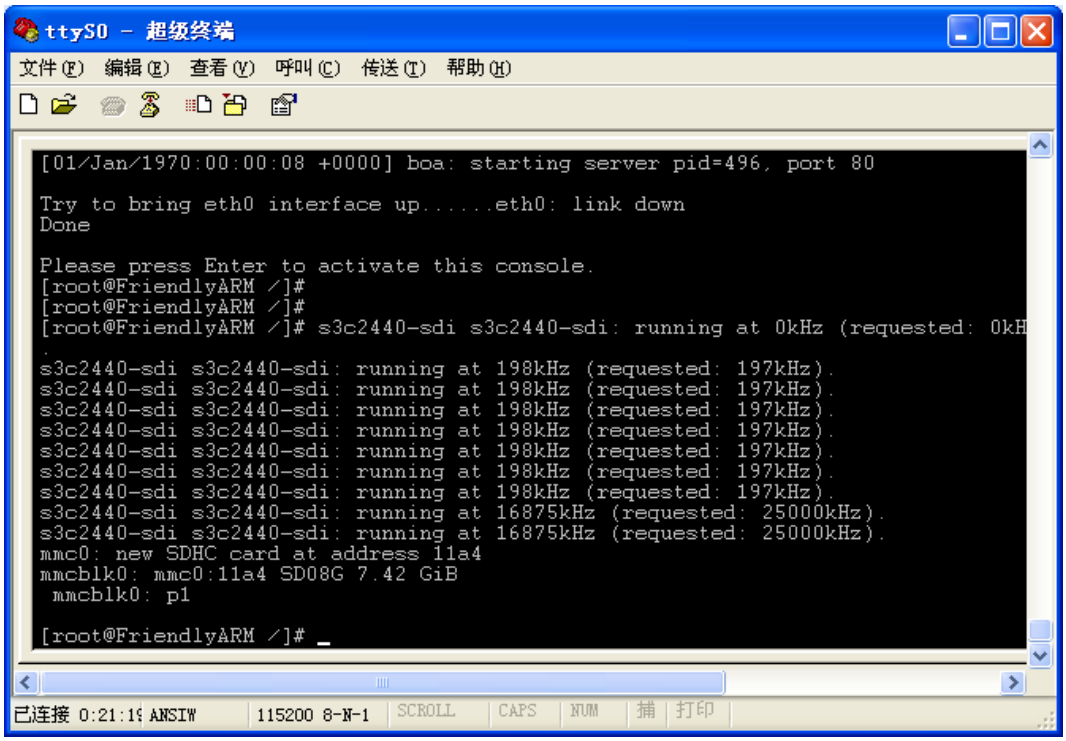


实际上优盘设备对应的设备名为**/dev/udisk**。进入/udisk 目录，就可以看到里面的文件了。注意：如果你的优盘无法识别，请检查一下它是不是 FAT32/VFAT 格式的

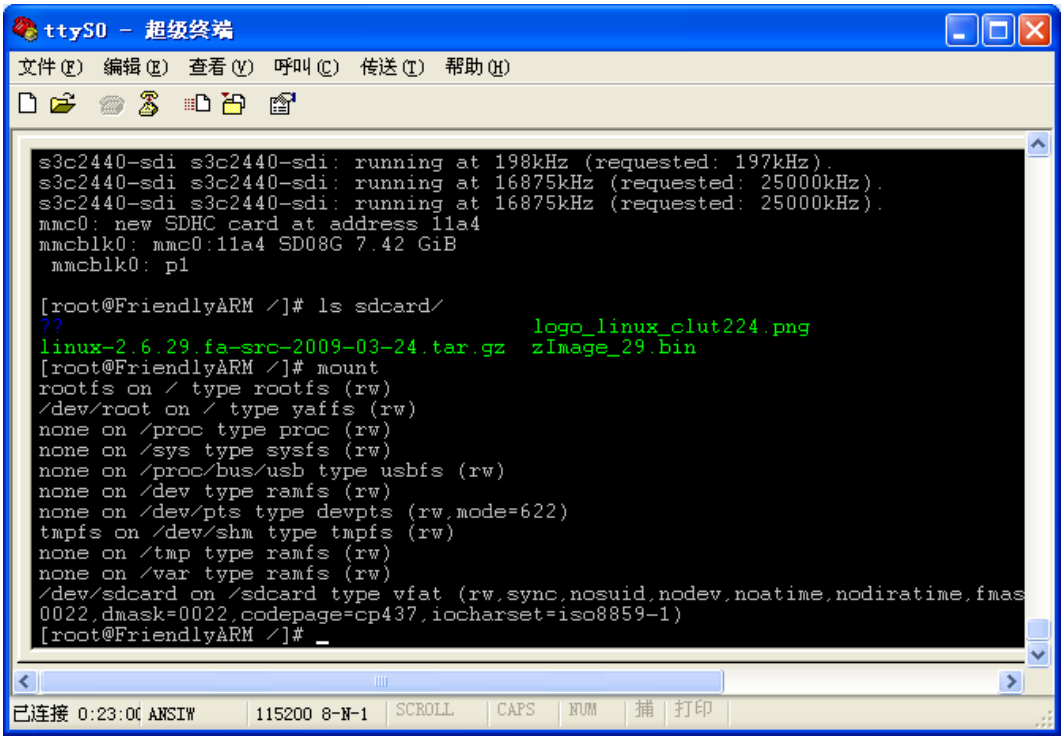


1.2.4 使用 SD 卡

和使用优盘一样，SD 卡也是自动识别挂载的，插入 SD 卡之后可以在串口看到如下信息：



系统会会自动创建/sdcard 目录，并把 SD 设备挂载到上面，如图：



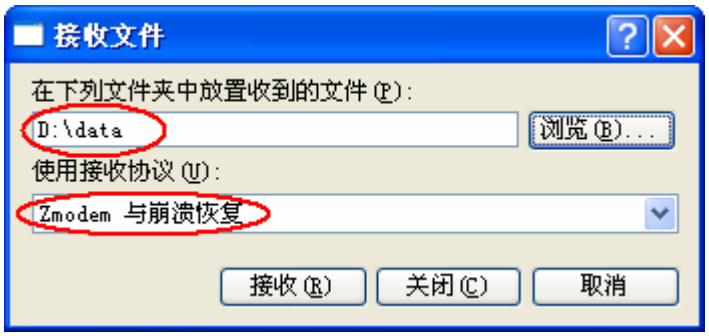
1.2.5 如何通过串口与 PC 互相传送文件

注意：使用 USB 转串口，有可能是不能顺利的，我们认为这和 USB 转串口线的质量和性能有关。

当通过串口终端登录系统之后，可以使用 **rz** 或者 **sz** 命令通过串口与 PC 互相传送文件，具体操作如下。

(1)使用 sz 向 PC 发送文件

在超级终端窗口中，点鼠标右键，在弹出的菜单中选择“接收文件”开始设置接收文件目录和协议，如图所示。



然后在终端的命令行输入“**sz /root/Documents/viva-la-vida.mp3**”命令，开始向 PC 传送位于“**/root/Documents/**”目录的 **viva-la-vida.mp3** 文件(或者其他文件，改一下路径和文件名就可以了)，因为该文件比较大，所以需要多等几分钟，发送完毕，系统会自动保存文件到您设置的目录里面，如图。

(2)使用 rz 命令下载文件到开发板

在串口终端输入“rz”命令，开始接收从 PC 传过来的文件。

然后在超级终端窗口中，点鼠标右键，在弹出的菜单中选择“发送文件”，设置好要发送的文件和使用的协议，如图所示，开始向开发板发送文件。

点“发送”，开发板开始接收文件，如图所示。

接送完毕，将会在当前目录下得到同样文件名的文件，您可以使用 md5sum 命令验证该文件是否和源文件相同。

1.2.6 控制板上的 LED

测试程序名称： led-player leds		备注
源代码文件名	led-player.c led.c	
源代码在光盘中的位置	解压 Linux\examples.tgz 可得	
开发板上对应的设备名	/dev/leds	
对应的内核驱动源代码	Linux-2.6.36/drivers/char/mini6410_leds.c	
其他：		

程序名称： leds.cgi		备注
源代码或者源代码包的名称	Leds.cgi	在开发板中
源代码或者源代码包的位置	位于开发板中的/www 目录中	
说明： leds.cgi 是一个 shell 脚本文件，它并不是二进制程序，该脚本通过 leds.html 被调用，其中使用的是最普通的网页设计技术。 解压光盘中的 root_default.tgz 也可以在其中的 www 目录得到 leds.cgi 和 leds.html 文件，它们都是脚本，本身就是源代码，使用任何文本编辑器(如 Windows 的“记事本”)都可以打开。		

说明：Led-player 和通过网页控制 LED 均为友善之臂早期为 SBC2410 开发的简易示例程序，因其硬件无关性，所以可以方便的移植到其他系统。目前市面上有的书籍，部分 2410/2440/6410 开发板厂商均采用了这个典型的管道应用示例。

(1) LED 服务器

开机进入系统后，将会自动运行运行一个 LED 服务程序(**/etc/rc.d/init.d/leds**)，它其实是调用了 **led-player** 的一个脚本，led-player 开始运行后，将会在/tmp 目录下创建一个 **led-control** 管道文件，向该管道发送不同的参数可以改变 led 的闪烁模式：

#echo 0 0.2 > /tmp/led-control

运行该命令后，4 个用户 led 将会以每个间隔 0.2 秒的时间运行跑马灯。

#echo 1 0.2 >/tmp/led-control

运行该命令后，4 个用户 led 将会以间隔 0.2 秒的时间运行累加器。

#/etc/rc.d/init.d/leds stop

运行该命令后，4 个用户 led 将会停止闪动。

#/etc/rc.d/init.d/leds start

运行该命令后，4 个用户 led 将会重新开始闪动。

(2)单独控制 LED

/bin/leds 是一个可以控制单个 led 的实用程序，要使用 leds 必须先停止 led-player，如下命令：

#/etc/rc.d/init.d/leds stop

该命令将停止 led-player 对 led 的操纵。led 的使用方法如下：

[root@fa /]# **led**

Usage: leds led_no 0|1

led_no 是要操作的 led(可为 0，1，2，3)，0 和 1 分别代表关闭和点亮。

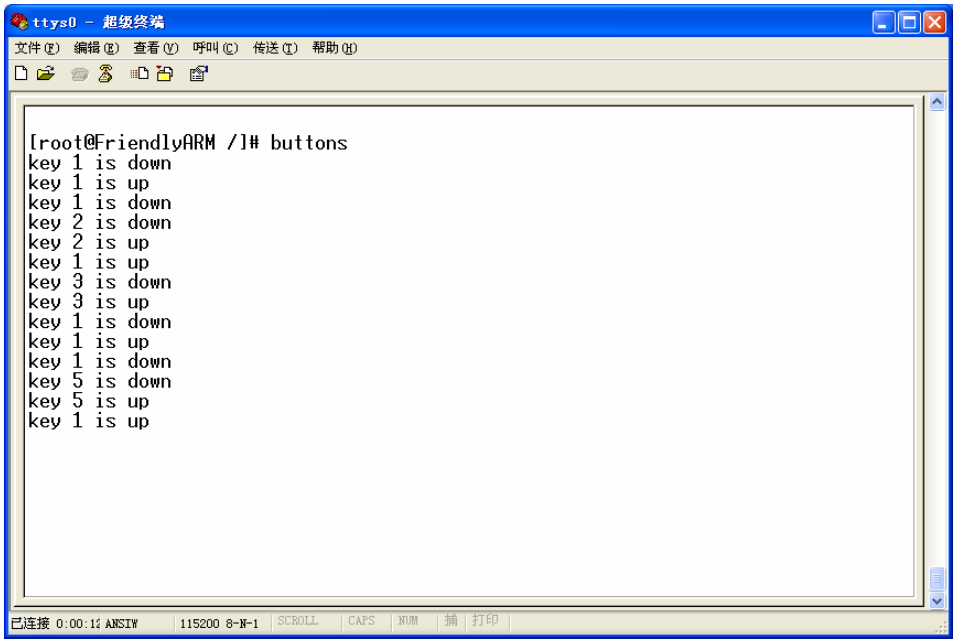
#led 2 1

将点亮 LED3

1.2.7 测试板上的按键

测试程序名称： buttons		备注
测试程序源代码文件名	Buttons_test.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
开发板上对应的设备名	/dev/buttons	
对应的内核驱动源代码	Linux-2.6.36/drivers/char/mini6410_buttons.c	
其他：		

在命令行输入“**buttons**”命令，然后按开发板上的按键，可以显示对应的键值，如图



1.2.8 串口测试

说明：armcomtest 是友善之臂为了方便测试而开发的 linux 下的简易实用串口终端程序，它使用标准的系统调用，和硬件无关。一般 Linux 系统系统启动后，串口 0,1,2 对应的设备名分别为**/dev/ttySAC0,1,2,3**

测试串口 2 需要借助另一台带有串口的 PC，使用我们提供的串口线和扩展小板(选购配件)，连接好 COM2 和另一台 PC 的串口，并如前所述设置该 PC 的超级终端为波特率 115200，无流控制，其他默认。

在命令行下输入：

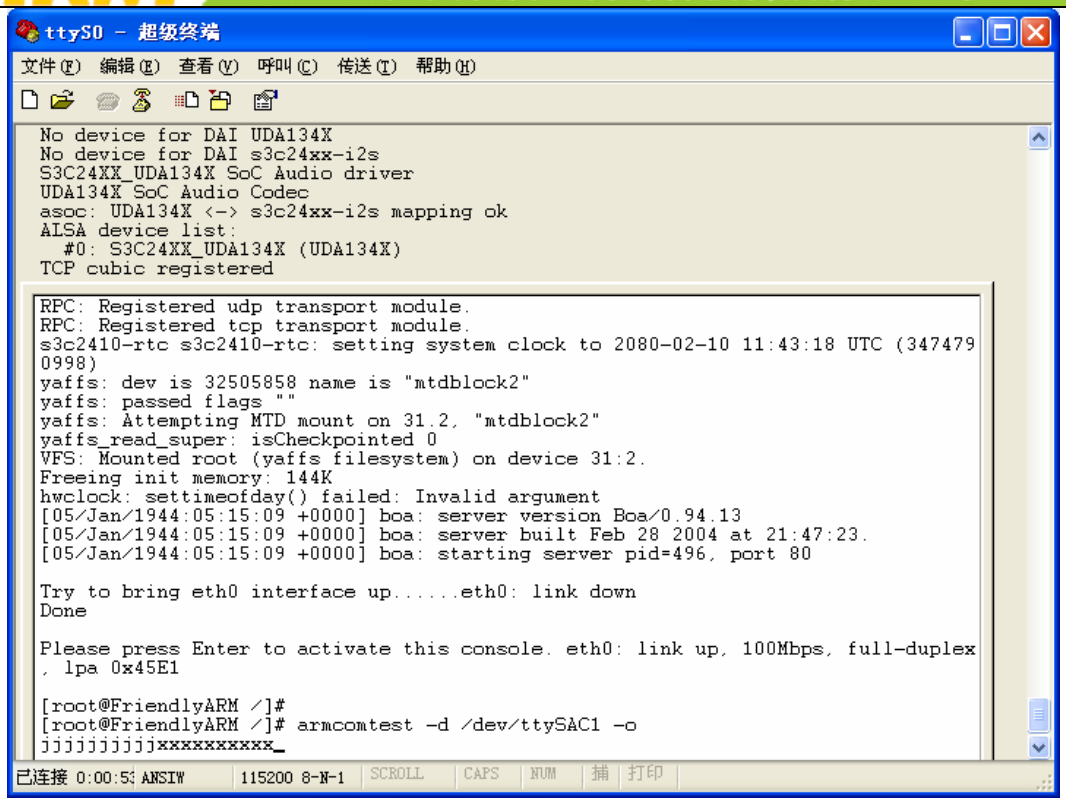
#armcomtest –d /dev/ttySAC1 -o

这时如果输入字符会在另一台 PC 的超级终端出现，反之亦然。

如果要测试串口 3，则需要连接扩展小板的 COM3，并在命令行输入：

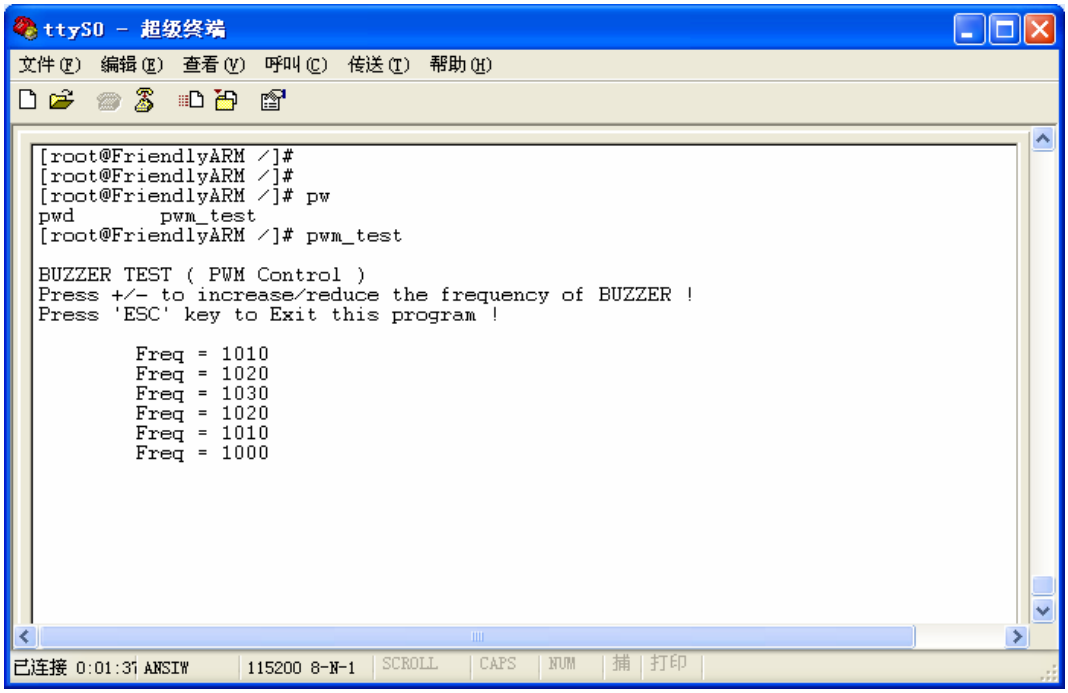
#armcomtest –d /dev/ttySAC2 -o

下面是测试时的界面：



1.2.9 测试蜂鸣器

在命令行种输入：pwm_test
可以听到蜂鸣器的发出的声音，按“+”或者“-”可以改变输出的频率，如图。
按 ESC 键中止该测试。



1.2.10 调节控制 LCD 背光

提示：
LCD 背光设备文件：/dev/backlight-1wire

采用一线触摸的 LCD 驱动板，还内置了背光调节电路及实现，背光的控制参数也是通过一线协议和主板进行传输的，它可以支持 127 级背光调节，当发送数字”0”给背光设备文件时，将会关闭背光：

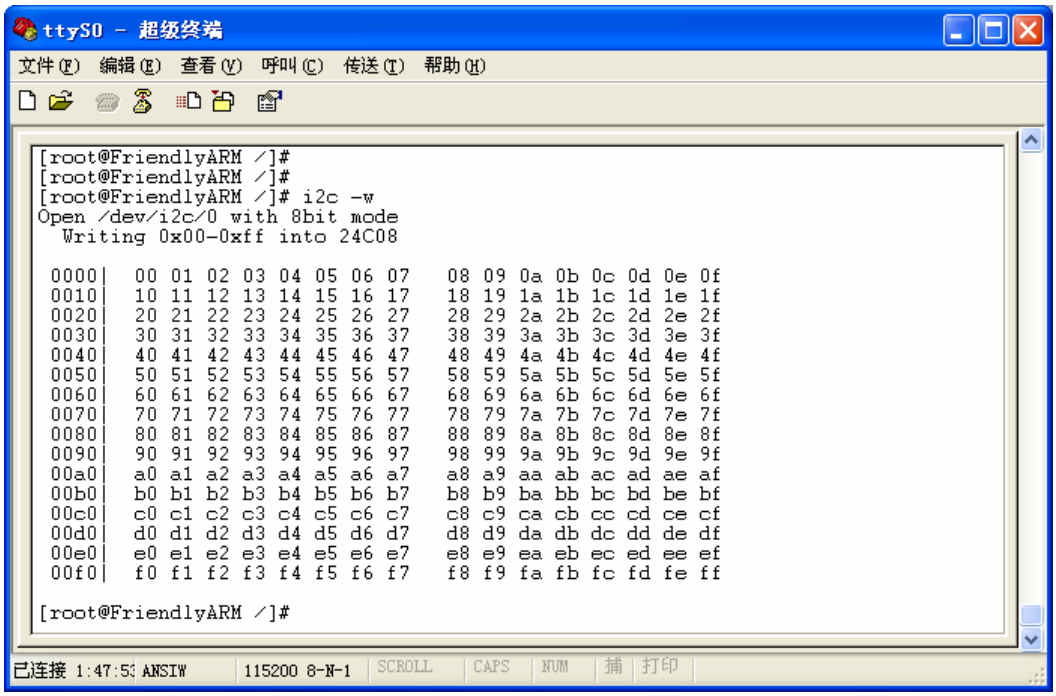
在命令行种输入：echo 0 > /dev/backlight 可以关闭 LCD 背光

当发送 1-127 给背光设备时，可以调节背光的亮度；其中 127 时为最亮，一般为 15 左右时，可以看到一些暗暗的画面，为 1-15 时，基本漆黑一片。当大于 127 时，将被视为 127 处理，也就是最亮。

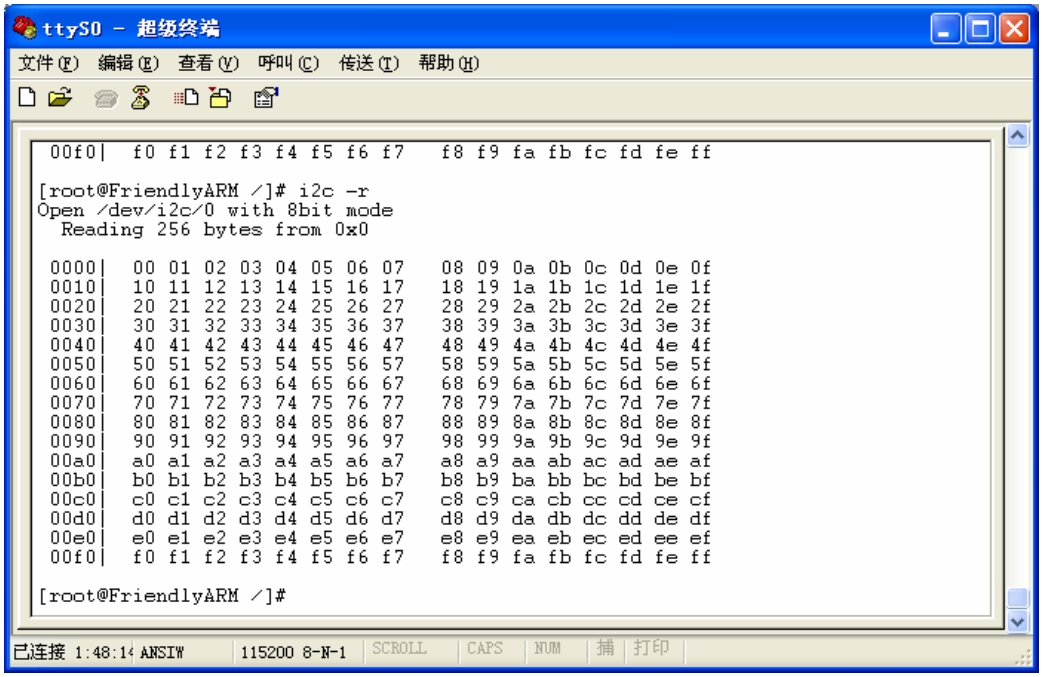
在命令行种输入：echo 15 > /dev/backlight 可以看到些许的背光。

1.2.11 测试 I2C－EEPROM

在命令行种输入：i2c –w 可以向板子的 24C08 器件中写入数据（0x00-0xff）



在命令行中输入：i2c –r 可以从板子的 24C08 器件中读出输出



1.2.12 AD 转换测试

测试程序名称： adc-test		备注
测试程序源代码文件名	Adc-test.c	

测试程序源代码位置	解压 linux\examples.tgz 可得	
开发板上对应的设备名	/dev/adc	
对应的内核驱动源代码	Linux-2.6.36/drivers/char/mini6410_adc.c	
其他：		

在命令行输入 adc-test 命令，可以进行 ADC 转换测试，调节开发板板上的可调电阻 W1，可以看到从串口终端输出的转换结果。

1.2.13 测试 TV-OUT

进入**/usr/bin** 目录，注意不是“/user/bin”，运行 **tv-test** 命令可以进行 TV-OUT 测试，程序将自动播放/usr/bin/TestVectors/wanted.264 文件，并输出至电视，如图

#cd /usr/bin

#tv-test

1.2.14 测试多媒体播放

见“Tiny6410 功能测试指南”1.3 章节

1.2.15 使用 USB 无线网卡或 SD WiFi

我们建议您使用图形界面程序，来设置和使用 WiFi 无线网卡，会更加方便，对于命令行方式的 WiFi 程序，我们将不再更新和维护。图形界面的设置见“Tiny6410 功能测试指南”第 1.12 章节。

注意：以下描述仅作之前的保留，不再更新。

为了方便在嵌入式 Linux 平台下使用 USB WiFi 和 SD WiFi 等无线网卡，我们曾基于在 Mini2440 开发板开发了一套命令行的 USB WiFi kits 工具程序，该工具集可以支持上千种型号的 USB 无线网卡(大部分的 USB 无线网卡使用的内部芯片是相同的)，现在我们已经把它移植到 6410 平台上，并且集成了 SD WiFi 驱动，

下面介绍一下该工具集的使用步骤：

该工具集包含了无线网卡驱动程序，和下面将要使用的三个实用命令行程序：

- scan-wifi – 用来扫描附近的无线网络

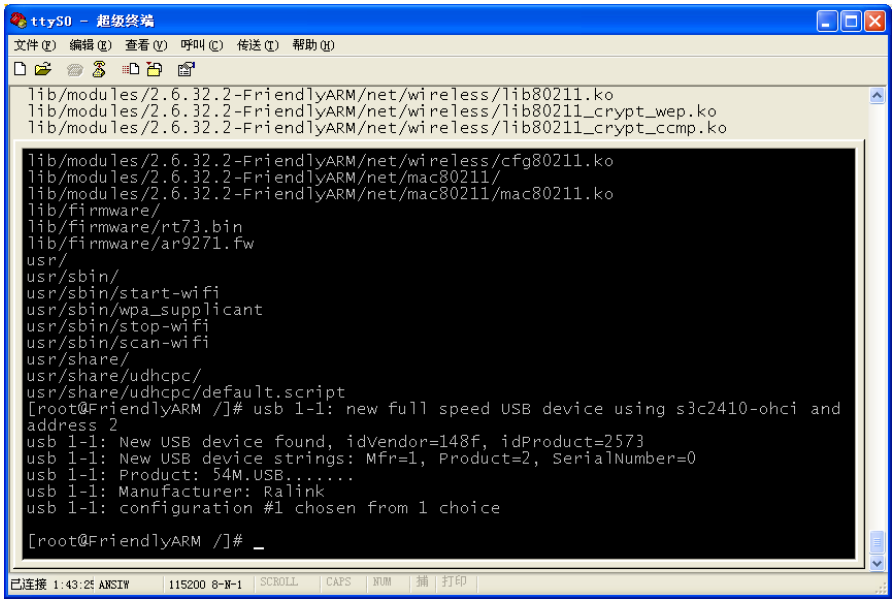
- start-wifi – 用来开启连接无线网络
- stop-wifi – 停止使用无线网络

这三个程序被安装在开发板的/usr/sbin 目录下

1. 扫描附近的无线网络

说明：以下示例使用 USB 无线网卡型号是：TL-WN321G+，SD-WiFi 模块的使用方式和此类似，就不再单独说明了。

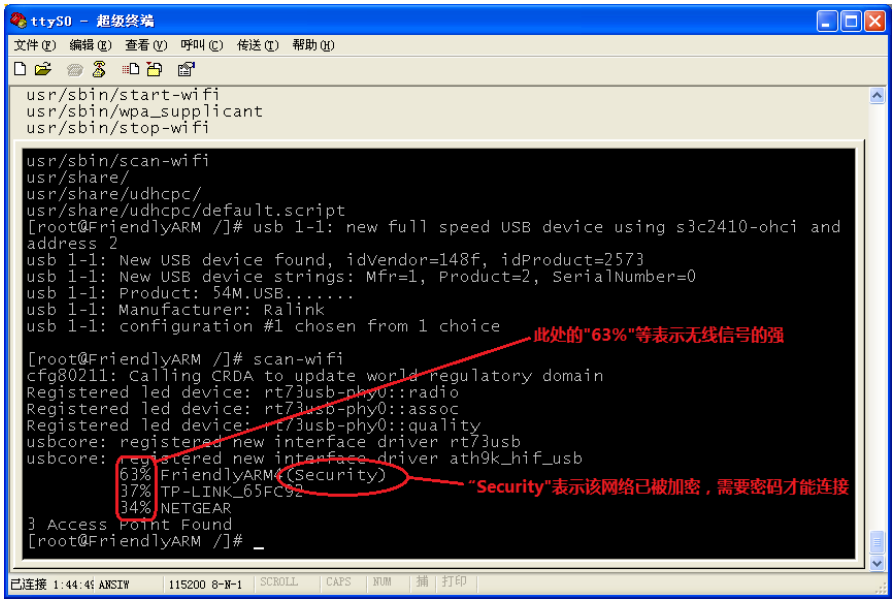
把 USB 无线网卡查到目标板上，会出现如下信息(网卡型号不同，信息也会不同)



执行扫描命令，以搜索附近的无线网络：

#scan-wifi

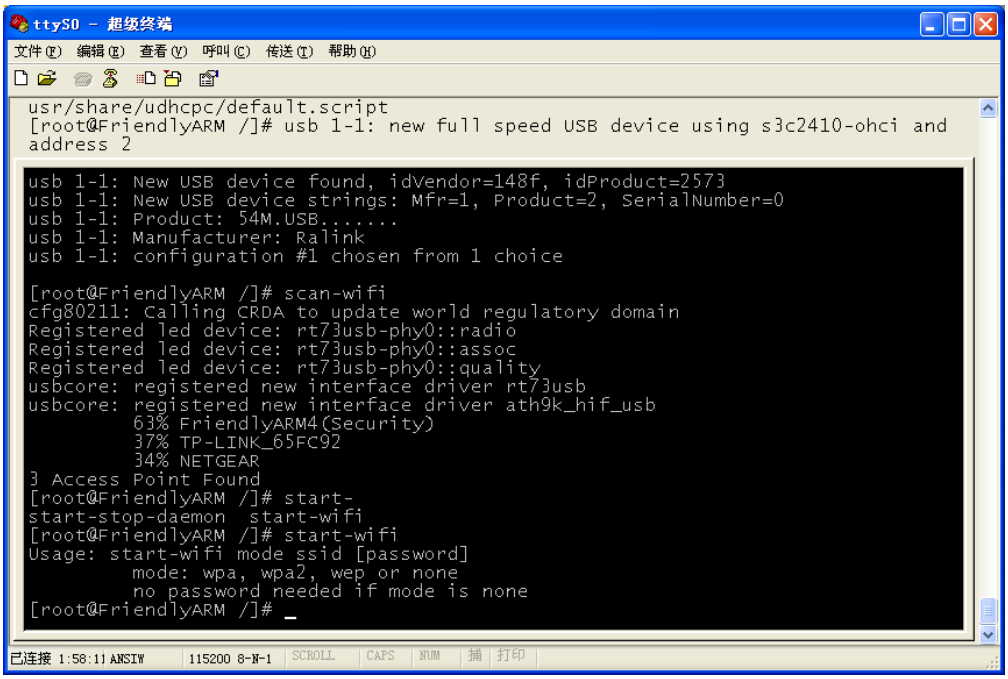
如图



可见已经搜索到 3 个无线网络，无线网名称前的“63%”表示信号的强弱，带有密码的安全网络会被标以“Security”。

2. 连接使用无线网

使用”start-wifi”命令可以自动连接到指定的无线网接入点，根据不同的无线网络特性，会有不同的参数，在命令行输入”start-wifi”可以看到如下提示信息：



其中,

mode – 表示无线网加密类型，可以为”wpa”, “wpa2”, “wep”或”none”, “none”表示没有不需要密码的无线网络。

ssid – 表示要连接的无线网络的名称，如上面的 “FriendlyARM4”, “NETGEAR”等，这个一般需要根据实际情况而定。

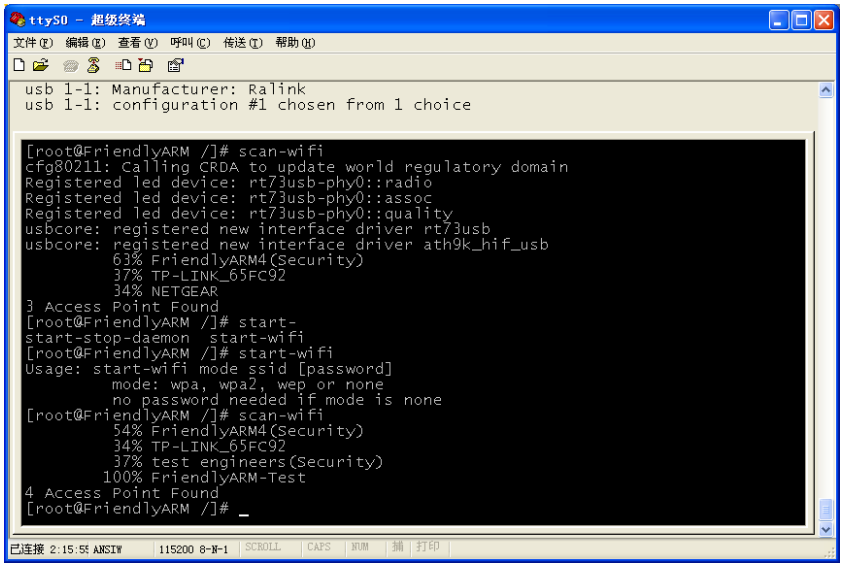
password – 表示加密的无线网所需的密码，将会以明文方式显示出来。

下面主要针对无安全加密和带安全加密的网络分别示例说明。

2.1 连接无需密码的开放无线网

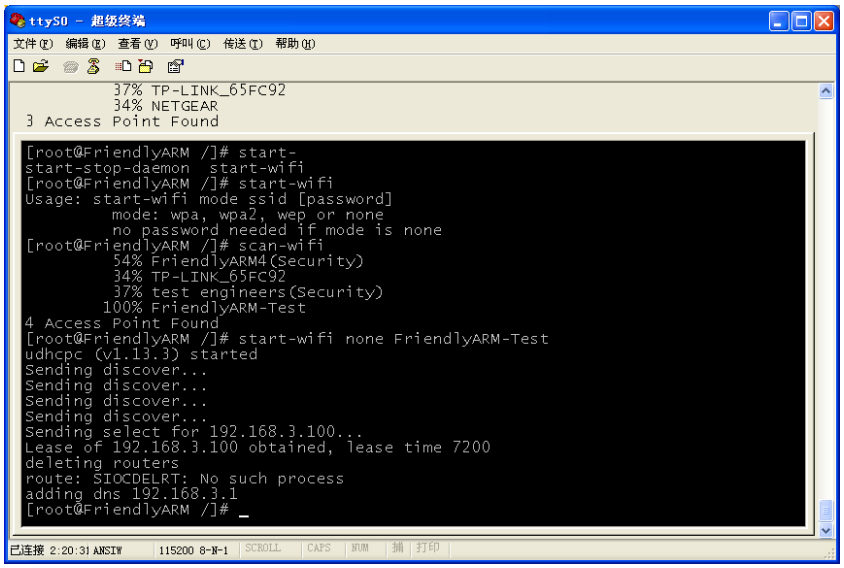
Step1

首先使用”scan-wifi”扫描查找附近的无线网络，结果如图，这里的“FriendlyARM-Test”是专门为测试而设立的一个无需密码的开放无线网接入点。

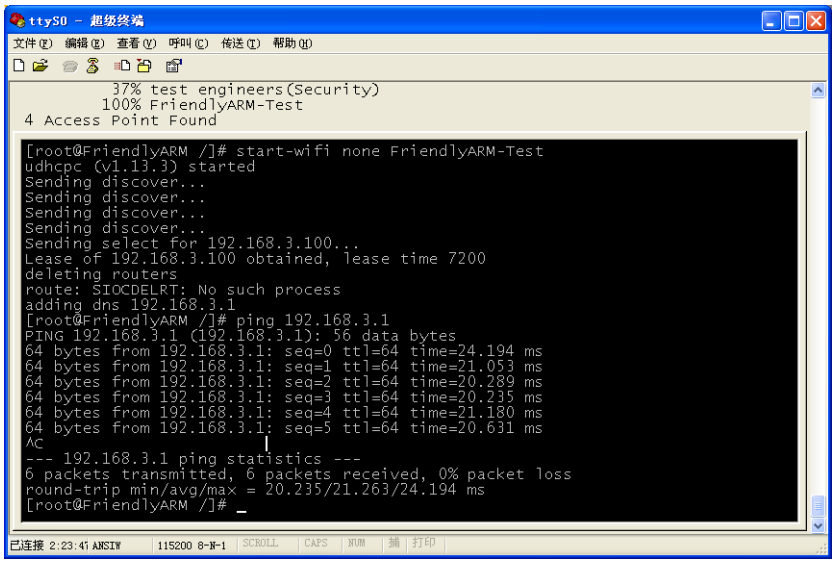


Step2

输入”start-wifi none FriendlyARM-Test”命令，开始自动连接该无线网络，如图。



稍等片刻，可以看到目标板已经自动分配到了 IP 地址：192.168.3.100，使用 ping 命令测试一下该网络连接，如图。



此时也可以在 PC 浏览器上输入开发板的 ip 地址：192.168.3.100，查看开发板中的 web 服务器，如图。

2.2 连接需要密码的安全无线网

连接使用带密码的无线网络的步骤和以上类似，只不过连接的时候需要事先知道无线网的加密类型和密码，如果你不知道加密类型，只能在”wpa”, “wpa2”, “wep”这三个中猜选了，具体步骤如下：

Step1

设置无线路由的安全模式，这里使用的无线路由器型号为：TL-WR740N，打开设置页面，如图

可以看到，此处有三种加密模式：

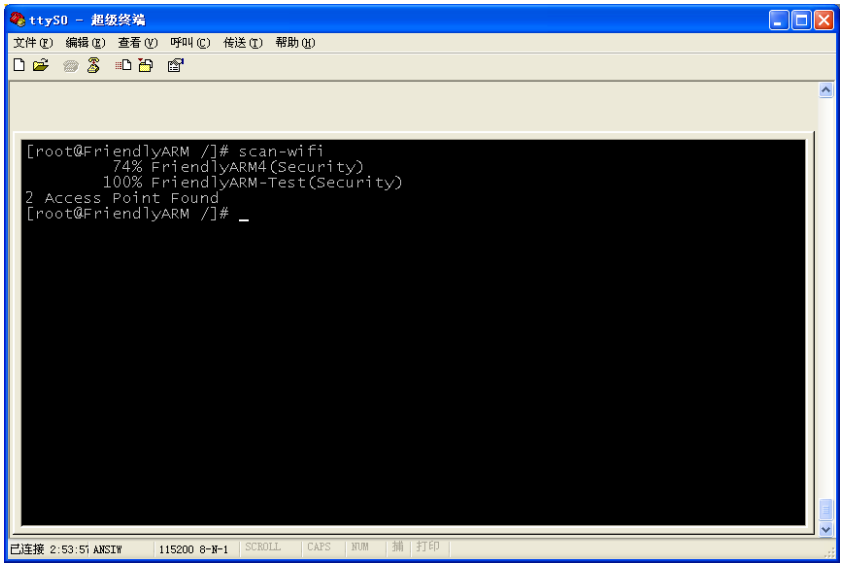
- WPA-PSK/WPA2-PSK
- WPA/WPA2
- WEP

我们选择的是第一种”WPA”，它是为个人而设的一种常见安全加密模式，在此设定密码为”test1234”，点保存，并重启启动路由器。

说明：关于如何设置无线路由器，我们在此并不作详细的介绍说明，大部分这种设备都有配套的使用说明书，并且每个厂家会有所不同，请根据实际情况自行设定。

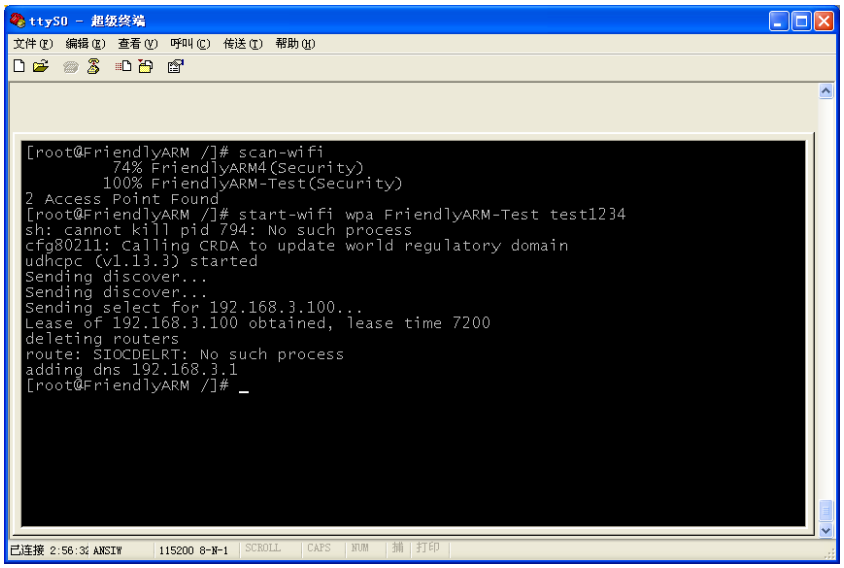
Step2

使用”scan-wifi”扫描查找附近的无线网络，结果如图，这里的“FriendlyARM-Test”是专门为测试而设立的一个无需密码的开放无线网接入点，可见，它是被加密的无线网络。

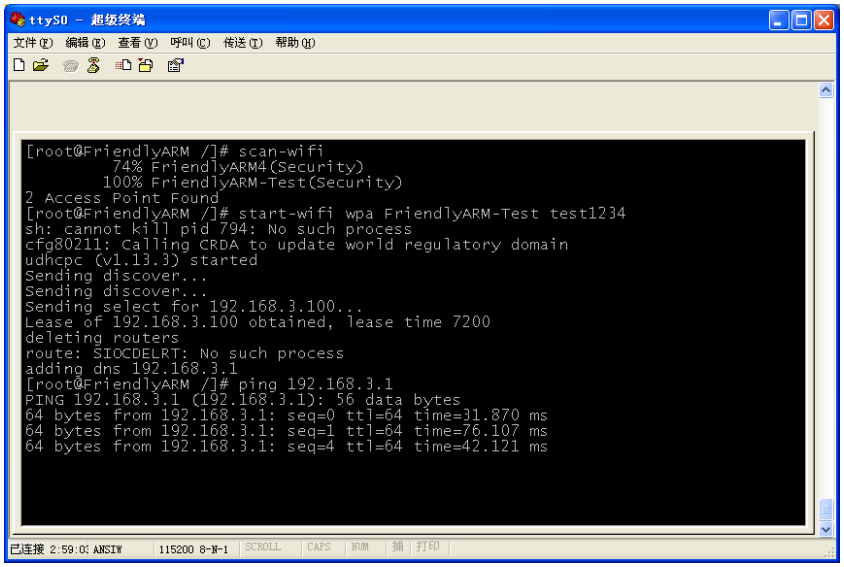


Step3

输入”start-wifi wpa FriendlyARM-Test test1234”命令，开始自动连接该无线网络，如图。



稍等片刻，可以看到目标板已经自动分配到了 IP 地址：192.168.3.100，使用 ping 命令测试一下该网络连接，如图。



此时也可以在 PC 浏览器上输入开发板的 ip 地址：192.168.3.100，查看开发板中的 web 服务器，如图。

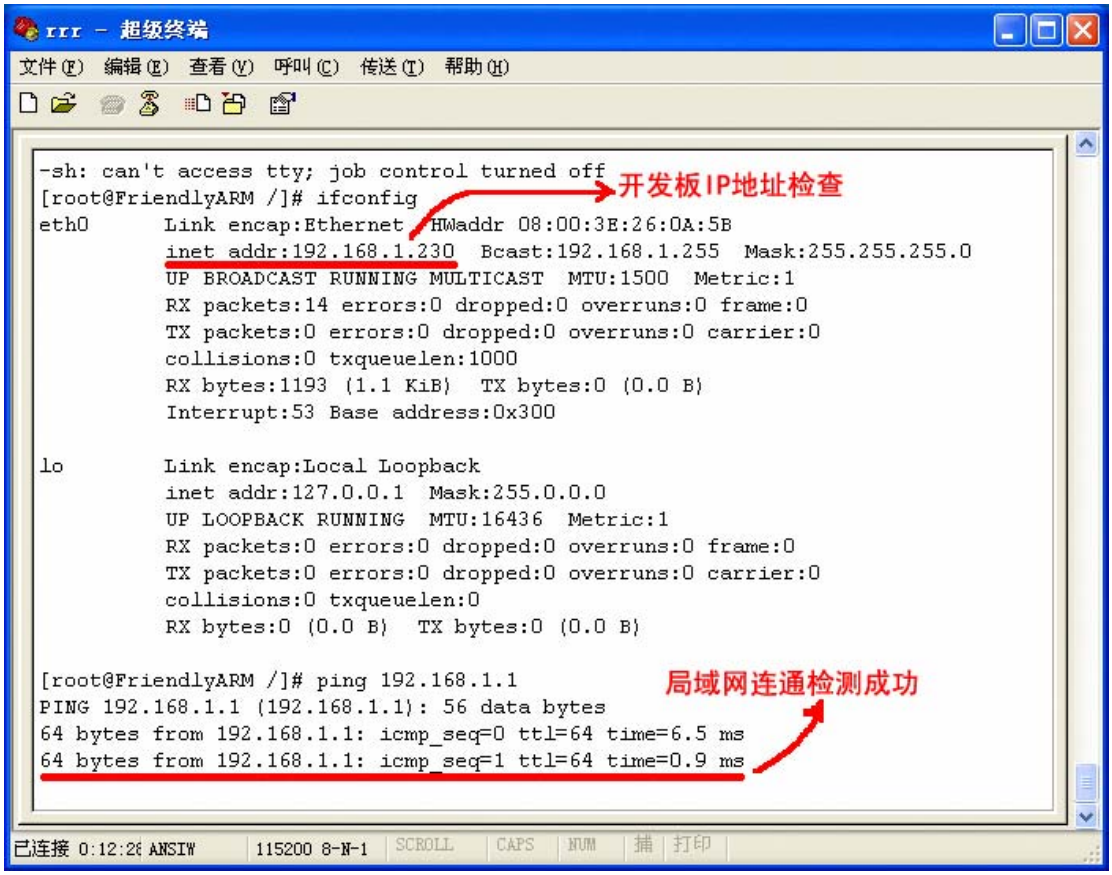
3. 断开 USB 无线网

要断开发板上的 USB WiFi 连接，可以使用在命令行输入“stop-wifi”命令，在此就不再截图说明了。

1.2.16 使用 telnet 上 bbs

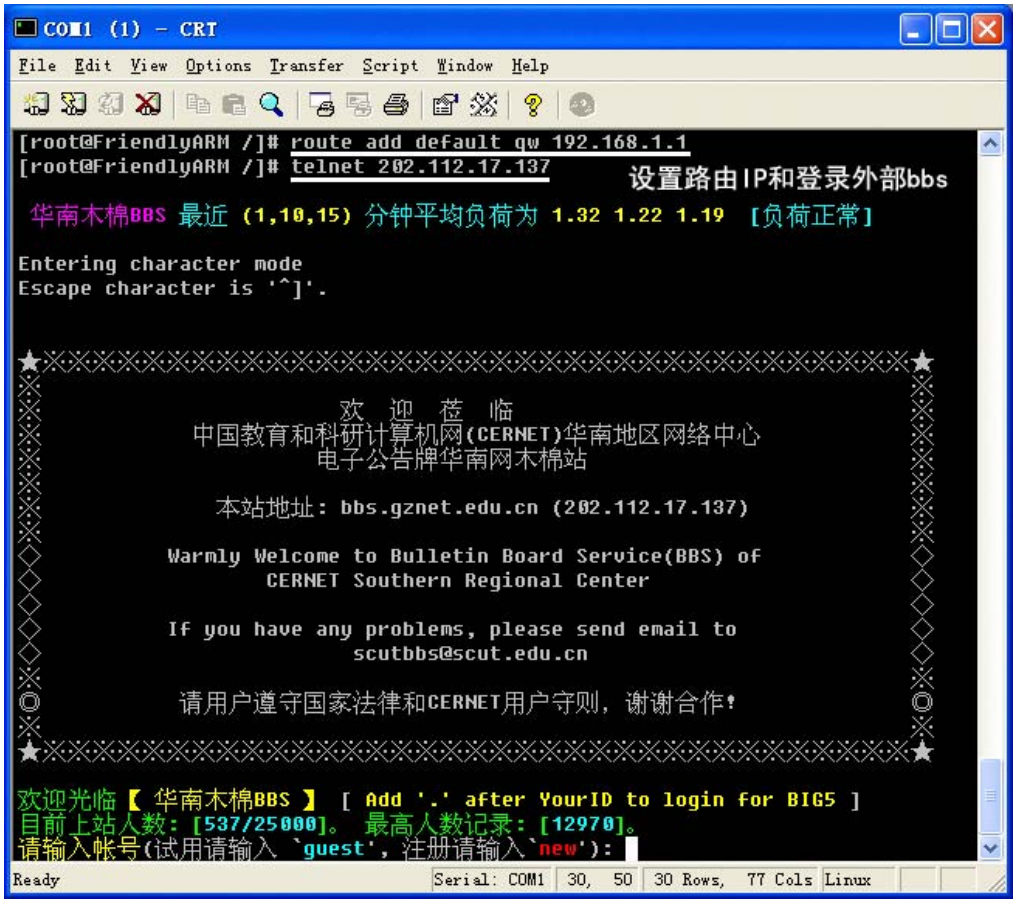
telnet 是一个经常被使用的远程登录工具，使用 telnet 功能，可以从开发板登录到其他提供了 telnet 服务器的主机，如果您接入开发板的网络可以上互联网，则可以通过 telnet 命令登录外部的 bbs。

首先，确认开发板的 IP 地址是否为 192.168.1.230，并且是否和局域网内其他主机相通，如图为成功的信息。



然后设置路由 IP：**route add default gw 192.168.1.1**

最后使用 telnet 命令登录您要登录的主机，在此登录的是华南木棉 bbs。



1.2.17 如何设置网络以访问互联网

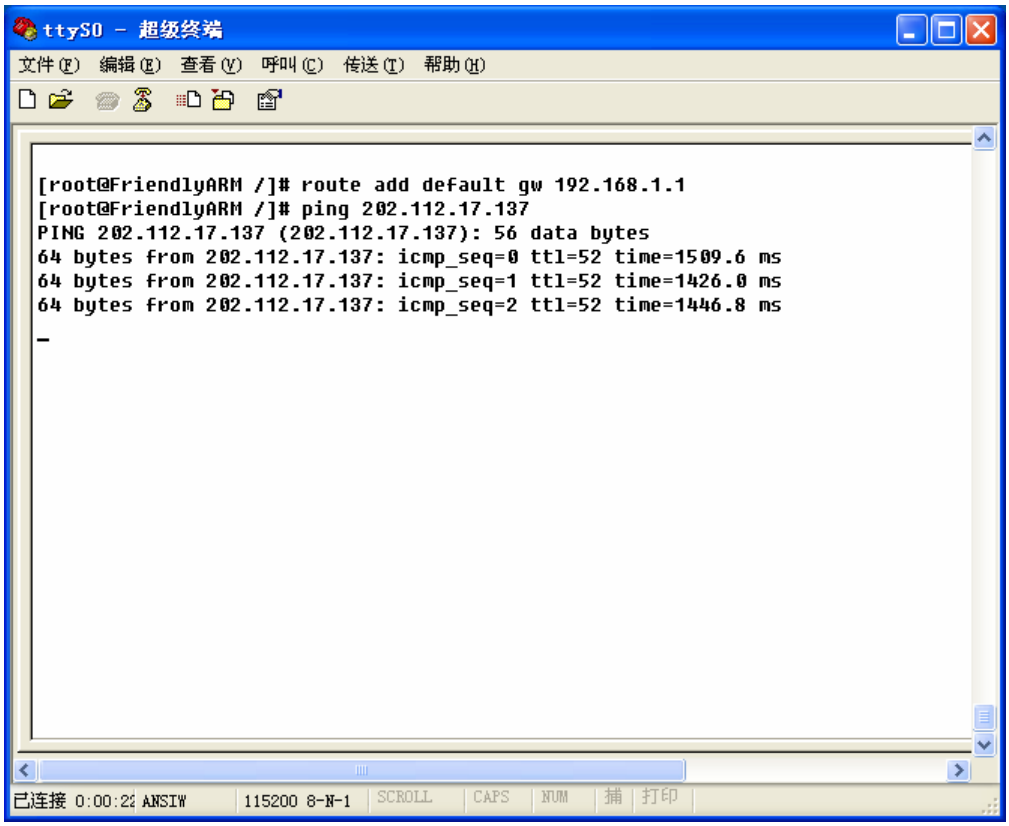
首先要确保你的网络环境可以正常登陆互联网，请记下你的网络环境所使用的网关 IP 地址，比如在我这里是 192.168.1.1，然后使用 route 进行设置：

 # route add default gw 192.168.1.1

这时你就可以直接访问互联网上的数字 IP 地址了，比如 ping 一下华南木棉的 BBS(其 IP 地址为 202.112.17.137)：

#ping 202.112.17.137

如图所示表示可以 ping 通外面的网络：



要能 ping 通外部网络的实名网址，还需要设置好域名解析服务器，先查看一下您当前网络所使用的 DNS 服务器 IP 地址(可以询问您的网络管理员)：

比如，我这里 DNS 服务器的 IP 为“202.96.128.86”，则在开发板中这样设置：

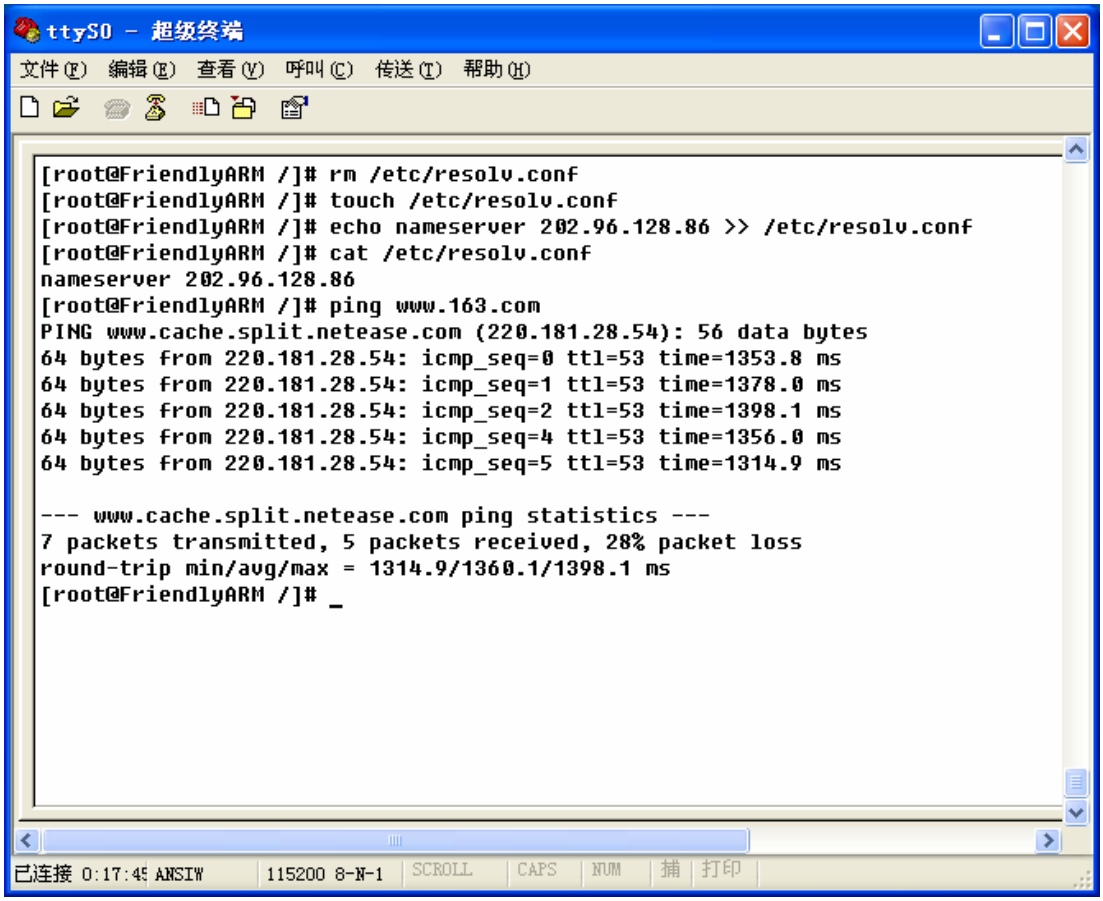
#rm /etc/resolv.conf ;首先删除以前的配置文件

#touch /etc/resolv.conf ； 重新生成一个 resolv.conf 文件

#echo nameserver 202.96.128.86 >> /etc/resolv.conf ;使用实际的 DNS 服务器 IP 配置 resolv.conf 文件

可以这里主要是修改/etc/resolv.conf 文件，当然你也可以直接使用 vi 进行修改。

全部过程如下图所示：

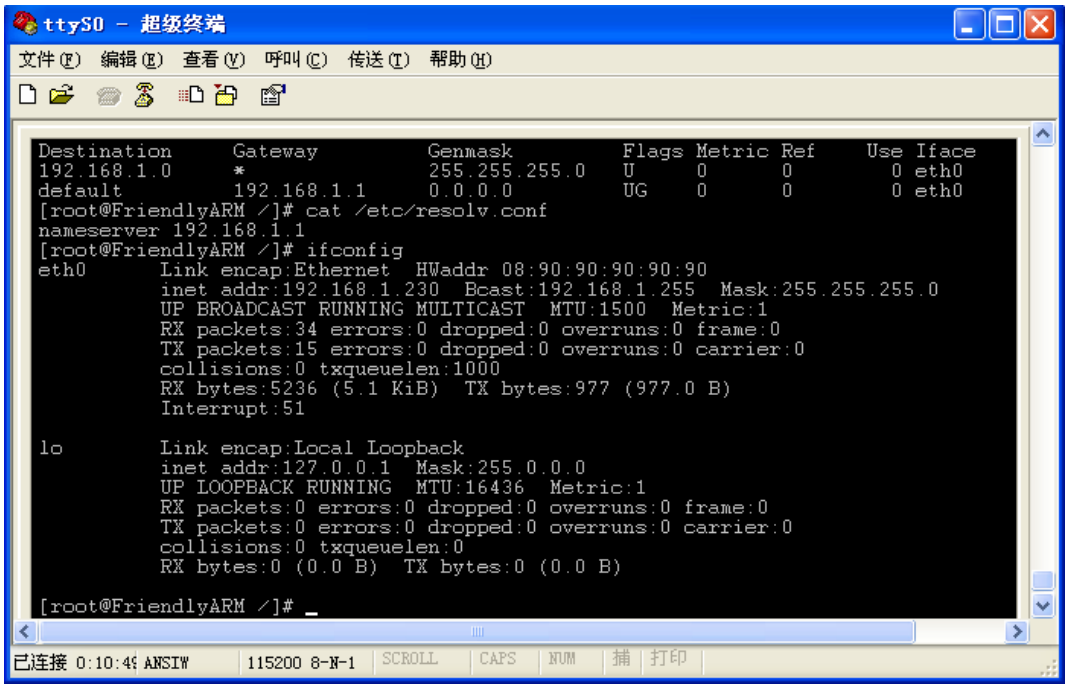


1.2.18 如何设置 MAC 地址

开发板中所使用的 MAC 地址是“软”性的，因此你可以通过 ifconfig 命令对它进行重置，以适应于在同一个网络环境中使用多片开发板的情况，具体操作如下：

首先使用 ifconfig 查看一下当前的 mac 地址，运行：

#ifconfig ;注意后面不要跟任何内容



可以看到当前的 mac 地址为“08：90：90：90：90：90”，这是在网卡驱动中默认的 mac 地址，它已经被写死到内核中，除非更改网卡的驱动源代码并重新编译得到新内核。要在运行的系统中动态更改 mac 地址，先关闭当前网络，并使用 ifconfig 重置 mac 地址：

#ifconfig eth0 down

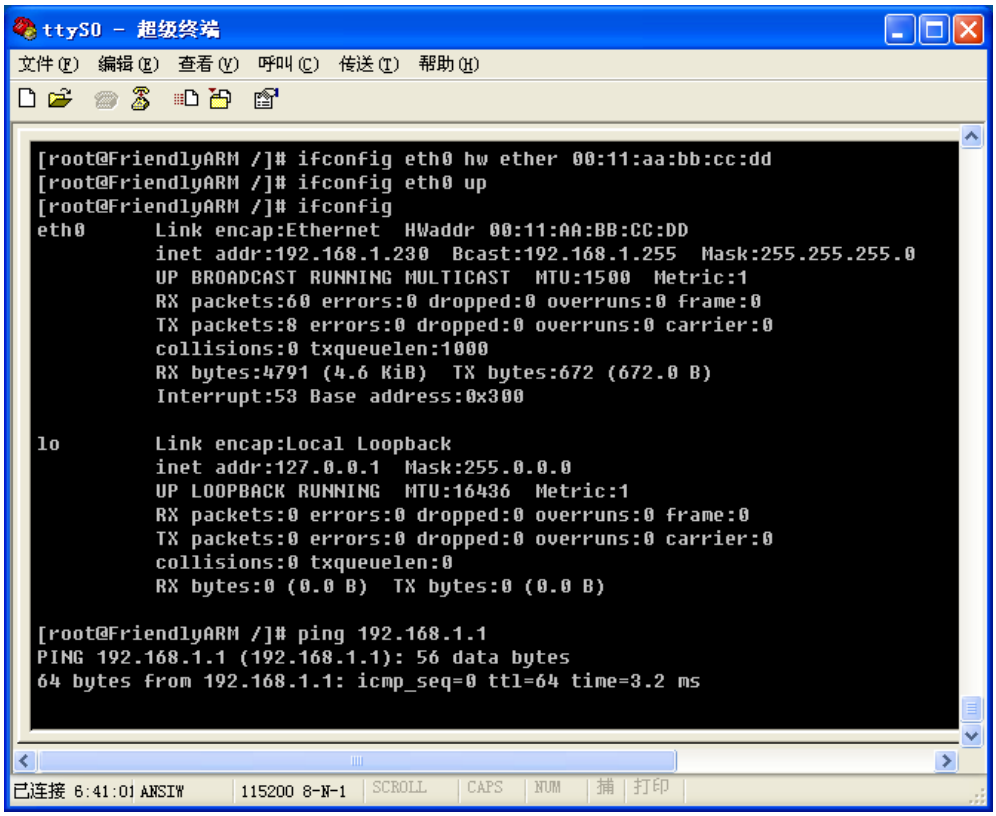
#ifconfig eth0 hw ether 00:11:AA:BB:CC:DD **;提示：a,b,c,d,e,f 可以为小写**

再开启网络，并使用 ifconfig 查看设置以后的 mac 地址，使用 ping 检验网络是否依然可通：

#ifconfig eth0 up

#ifconfig

#ping 192.168.1.1

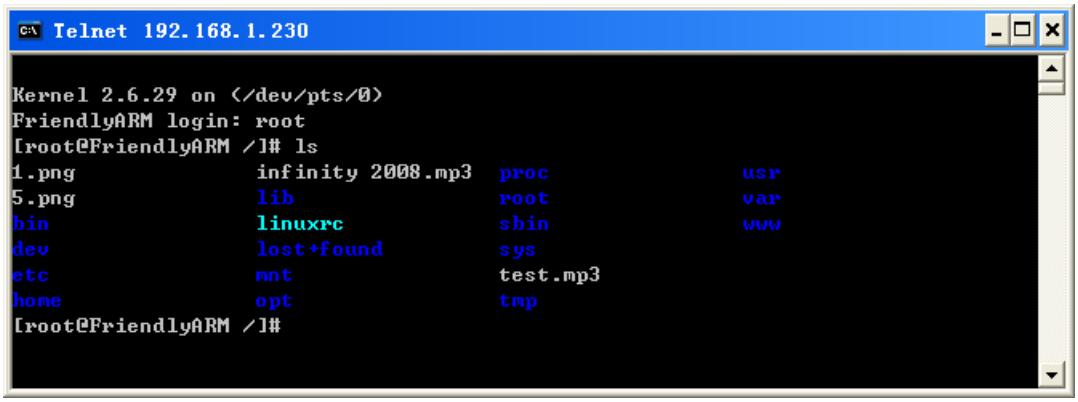


1.2.19 如何使用 Telnet 登录开发板

说明：6410 系统所用的部分程序和 2440 效果是相同的，以下截图使用了 mini2440 的截图，仅供参考，请以实际情况为准。

开发板开机正常运行后，其实已经启动了一个 Telnet 服务，因此用户也可以通过网络远程登录开发板。

在 Windows 的命令行窗口输入“**telnet 192.168.1.230**”，如图出现登录界面，输入“root”(不需要密码)进入系统。



1.2.22 使用 ftp 传递文件

说明：6410 系统所用的部分程序和 2440 效果是相同的，以下截图使用了 mini2440 的截图，仅供参考，请以实际情况为准。

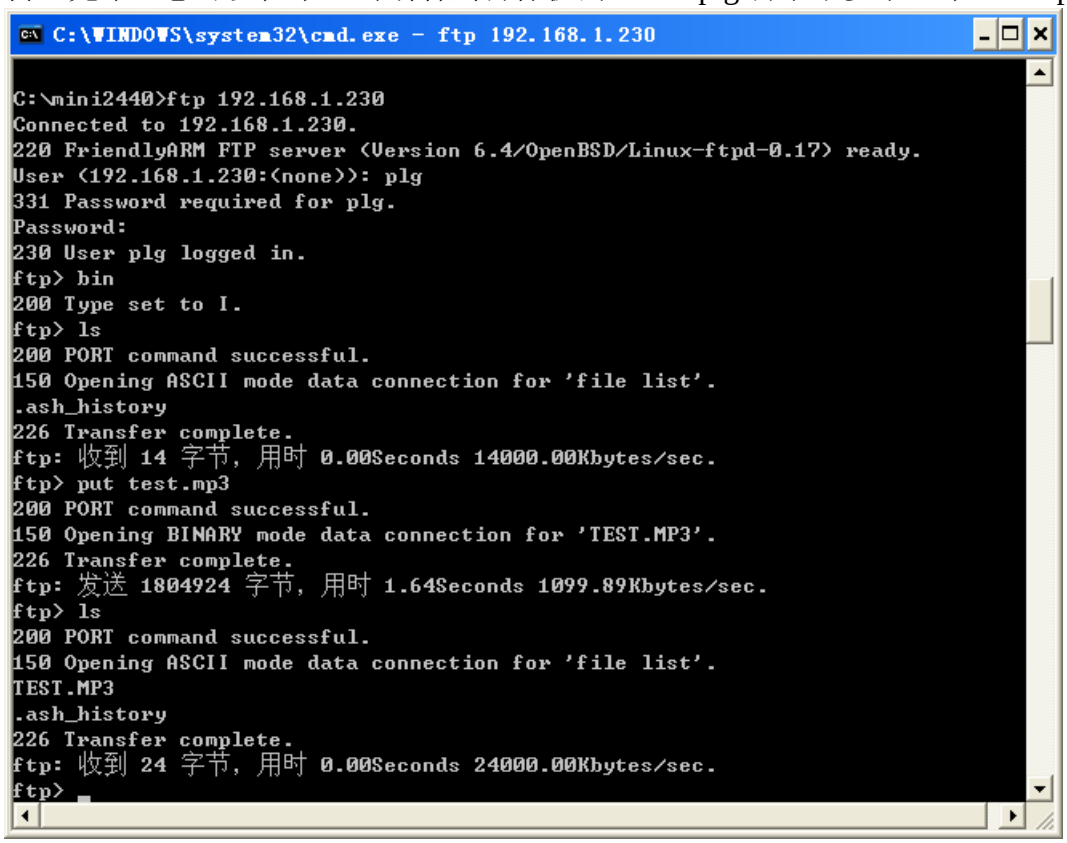
开发板开机正常运行后，其实已经启动了一个 Telnet 服务，因此用户也可

无论在 Linux 系统还是 windows 系统中，一般安装后都自带一个命令行的 ftp 命令程序，使用 ftp 可以登录远程的主机，并传递文件，这需要主机提供 ftp 服务和相应的权限；本开发板不仅带有 ftp 命令，还在开机时启动了 ftp 服务。为了方便测试，我们可以从 PC 机的命令行窗口登录开发板，并向开发板传递文件。

注意：请确保您执行 ftp 所在的目录有需要上传的文件，这里是 test.mp3

说明：登录开发板的 ftp 帐号为：plg 密码为：plg

传送完毕，您可以在串口终端看到目标板的/home/plg 目录下多了一个 test.mp3 文件。



1.2.23 通过网页控制板上的 LED

说明：6410 系统所用的部分程序和 2440 效果是相同的，以下截图使用了 mini2440 的截图，仅供参考，请以实际情况为准。

在 web server 测试页面中点“网络控制 LED 测试”项，会出现 LED 测试控制页面，如图

您可以使用网页中的各个测试项目进行测试，其中的“LED 测试”将会通过 CGI 程序来控制板上的 LED 灯，其中包括 2 种方式的显示类型和三种不同的显示速度。

如果要停止 web 服务器，则在命令提示符下输入以下命令：

#/etc/rc.d/init.d/httpd stop

要重新启动则输入：

#/etc/rc.d/init.d/httpd start

1.2.24 如何挂接使用网络文件系统 NFS

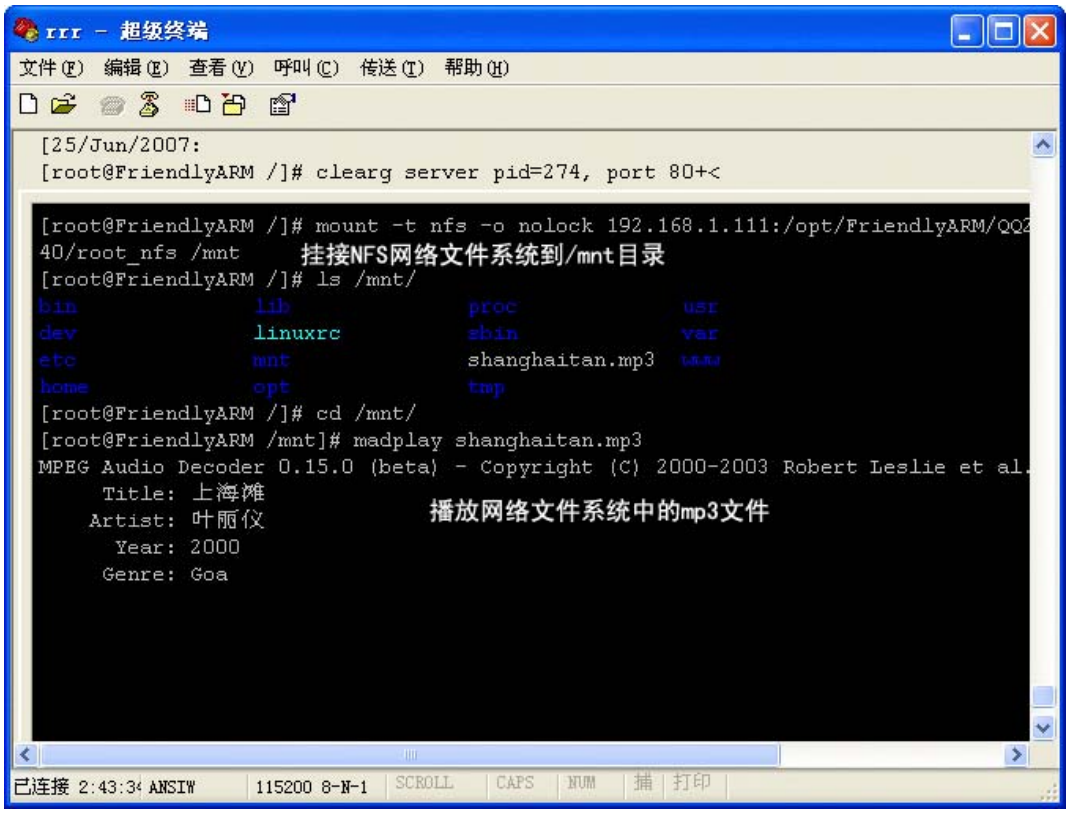
在进行该测试之前，请先在 PC 端搭建好 NFS 服务器系统，然后在命令行输入以下命令（假定服务器的 IP 地址为 192.168.1.111）：

```
#mount –t nfs –o nolock 192.168.1.111:/opt/FriendlyARM/mini6410/linux/rootfs_qtopia_qt4    /mnt
```

挂接成功，您就可以进入/mnt 目录进行操作了，如下图所示。

取消挂接的命令如下：

#umount /mnt



1.2.25 设置并保存系统实时时钟

Linux 中更改时间的方法一般使用 **date** 命令，为了把 S3C2440 内部带的时钟与 linux 系统时钟同步，一般使用 **hwclock** 命令，下面是它们的使用方法：

(1) **date -s 042916352007** #设置时间为 2007-04-29 16:34

(2) **hwclock -w** #把刚刚设置的时间存入 S3C2440 内部的 RTC

(3).开机时使用 **hwclock -s** 命令可以恢复 linux 系统时钟为 RTC，一般把该语句放入 **/etc/init.d/rcS** 文件自动执行。

注意：我们提供的系统已经把 hwclock –s 命令写入 rcS 文件。

1.2.26 如何掉电保存数据到 Flash

由于本系统采用了可读写文件系统 yaffs2(在嵌入式系统中，专门管理 Flash 存储器的一种文件系统)，因此可以很方便的动态保存数据，掉电后不会丢失。开机后在串口终端运行以下命令：

#cp / shanghaitan.mp3 /home/plg

此时将在/home/fa 目录下复制一个同样的文件，然后关机，重新开启系统，可以查看到/home/plg 目录下的文件依然存在。

1.2.27 设置开机自动运行程序

借助启动脚本可以设置各种程序开机后自动运行，也可以设置其他系统设置，这有点类似于 Windows 系统中的 Autobat 自动批处理文件，启动脚本的位于板子的/etc/init.d/rcS，内容如下(实际内容可能与此不完全一致)：

```
#! /bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

#
#       Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#
trap ":" INT QUIT TSTP
/bin/hostname FriendlyARM

[ -e /proc/1 ]       || /bin/mount -n -t proc    none /proc
[ -e /sys/class ] || /bin/mount -n -t sysfs none /sys
[ -e /dev/tty ]     || /bin/mount       -t ramfs none /dev
/bin/mount -n -t usbfs none /proc/bus/usb

echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp
```

```
/sbin/hwclock -s

syslogd
/etc/rc.d/init.d/netd start
echo "                                    " > /dev/tty1
echo "Starting networking..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/httpd start
echo "                                    " > /dev/tty1
echo "Starting web server..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/leds start
echo "                                    " > /dev/tty1
echo "Starting leds service..." > /dev/tty1
echo "                                    "
sleep 1

echo "                                    " > /dev/tty1
/etc/rc.d/init.d/alsaconf start
echo "Loading sound card config..." > /dev/tty1
echo "                                    "

/sbin/ifconfig lo 127.0.0.1
/etc/init.d/ifconfig-eth0

/bin/qtopia &
echo "                                                " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
```

1.2.28 如何使用命令进行屏幕截图

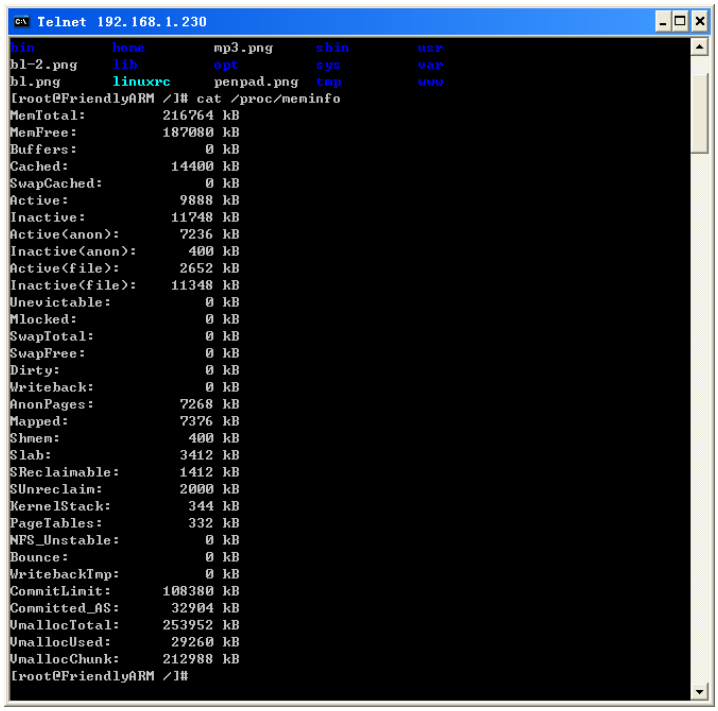
使用 snapshot 命令可以对当前的 LCD 显示进行截图，并保存为 png 格式的图片。
#snapshot pic.png
执行该命令将把当前 LCD 显示进行抓图，并保存为 pic.png 文件。

1.2.29 查看开发板内存信息

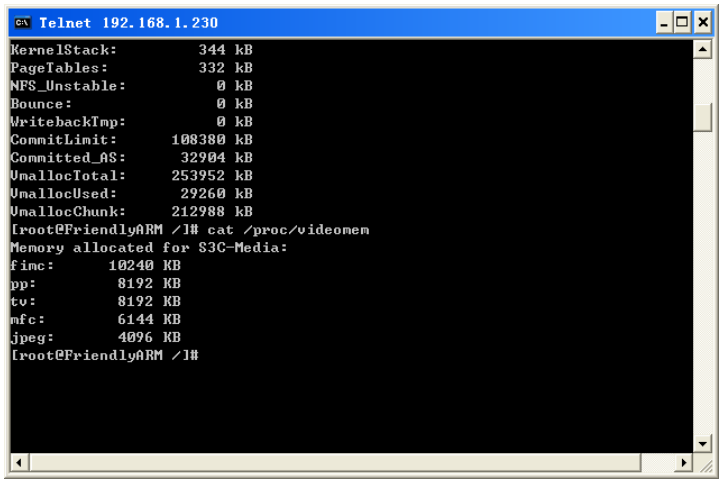
本开发板配备了 256M DDR RAM，但有用户反映在系统中只查看到大约 68M 总内存，

这是因为多媒体驱动“吃掉”了不少内存空间，下面我们对此简单说明一下：

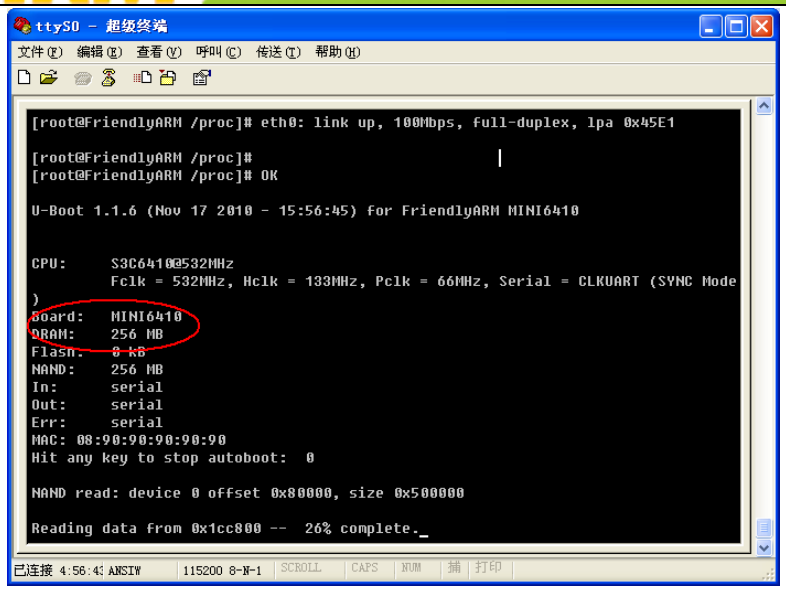
一般我们使用“cat /proc/meminfo”命令系统内存信息，该信息仅说明了整个 Linux 软件系统所能分配到的内存总量，它的结果如下图所示，总共为 216M 左右。



实际上，6410 内部的多媒体协处理器也会用到一些内存，运行“cat /prov/videomem”命令可以看到详细的内存分配，如图：



在 u-boot 启动系统之前，也可以查看到实际的内存大小，如图：



1.3 安装并设置 Fedora9

本小节从在虚拟机/PC 机上安装 Fedora 9.0 开始，详细介绍了如何建立 Linux 开发环境。

我们的软件开发和测试全部基于Fedora9 平台做开发，所有的配置和编译脚本也基于此平台，我们没有在其他平台上测试过。如果你对Linux开发很熟悉，相信你会根据错误提示逐步找到原因并解决，它们一般是你选用的平台缺少了某些库文件或者工具等原因造成的；否则，我们建议初学者使用和我们一致的平台，即Fedora 9(全称为: Fedora-9-i386-DVD.iso)，你可以在其官方网站下载(ftp://download.fedora.redhat.com/pub/fedora/linux/releases/9/Fedora/i386/iso/Fedora-9-i386-DVD.iso ，不保证长期有效)，也可以在其他地方获取，它们都是一样的，安装时请务必参考我们手册提供的步骤，这是我们经过严格测试的，以免遗漏一些开发时所需要的组件。

Linux 的发行版本众多，我们无法为此一一编写文档解释安装方法，请谅解。

我们为什么选择 Fedora 9:

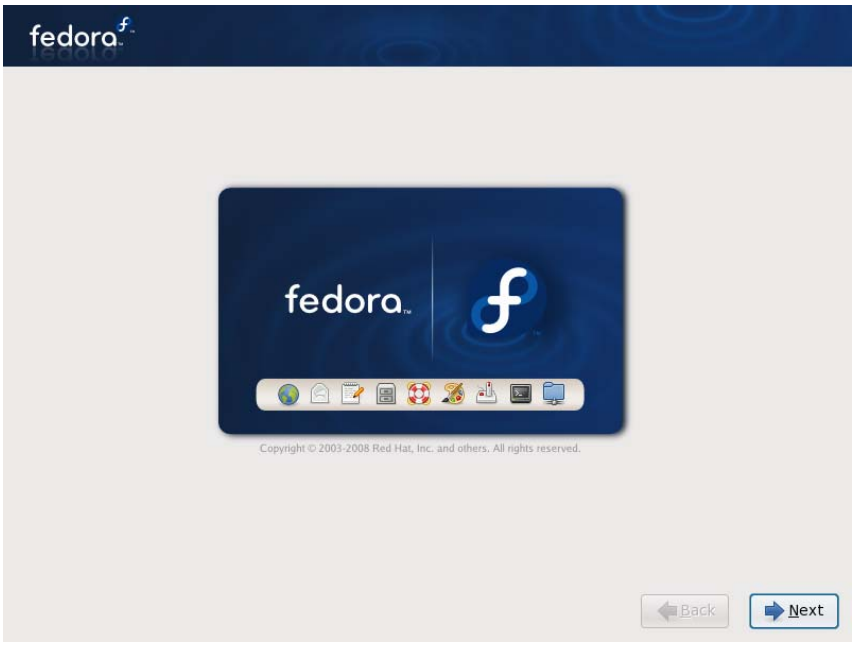
根据我们的测试，Fedora 9 经过比较简单的安装和设置，依然可以使用 root 用户登录(大多数开发均需要此用户权限)，Fedora 10 及其以后的版本则需要经过稍微复杂的设置才能使用 root，这不利于不了解 Linux 的初学者，Fedora 8 及其以前的版本则相对老了一些。并且按照我们手册提供的步骤安装 Fedora 9，可以比较完美配合我们提供的开发软件包，不再需要其他补丁之类的繁琐设置(ubuntu 就需要经常这样更新设置)，因此我们认为 Fedora 9 是最适合初学者的开发平台。

1.3.1 图解安装 Fedora 9.0

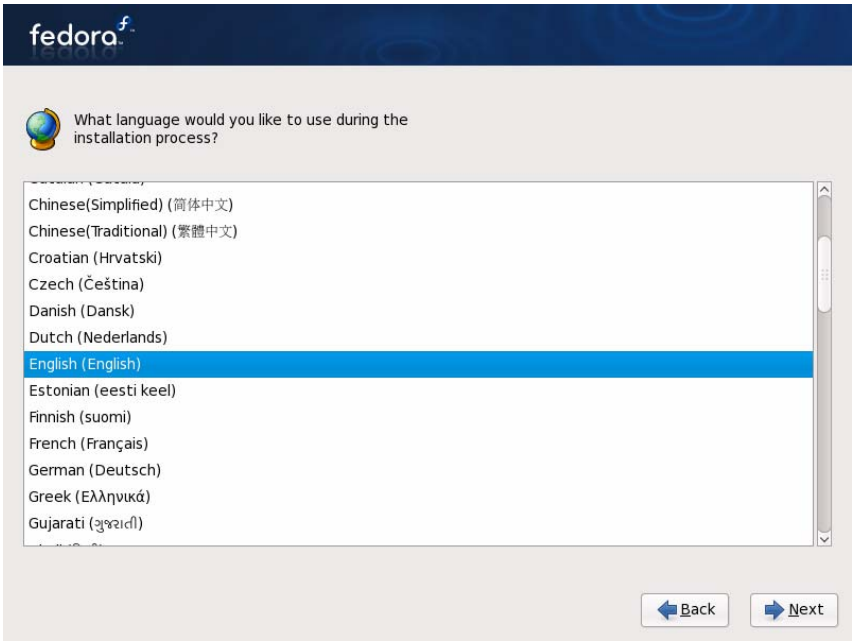
Step1： 将的安装光盘放到光驱中，将 BIOS 改为从光盘启动，启动后系统将会出现如下界面，按回车继续。

Step2：然后进入下一步，检查安装盘，一般不需要检测，所以选择了 Skip（跳过）

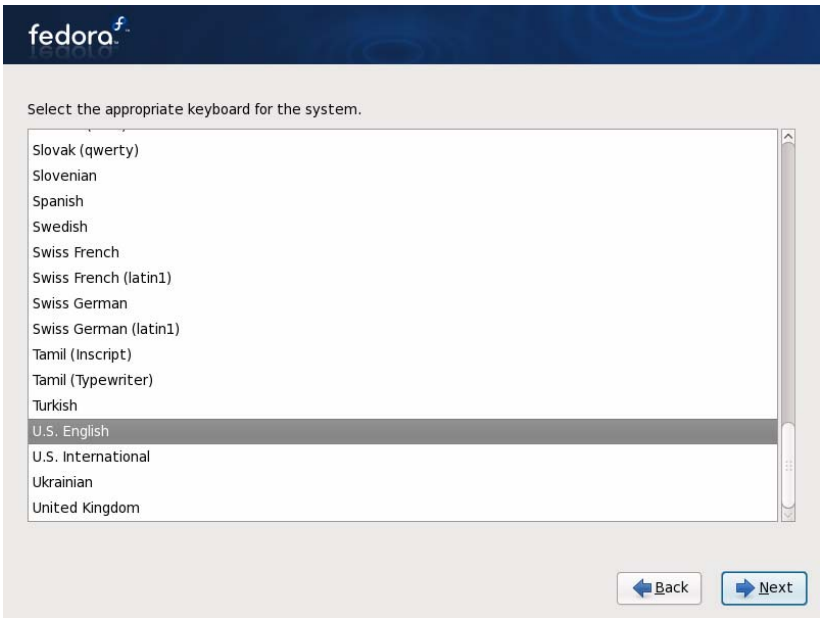
Step3：过一会儿就进入安装图形化画面，点击 Next 即可.



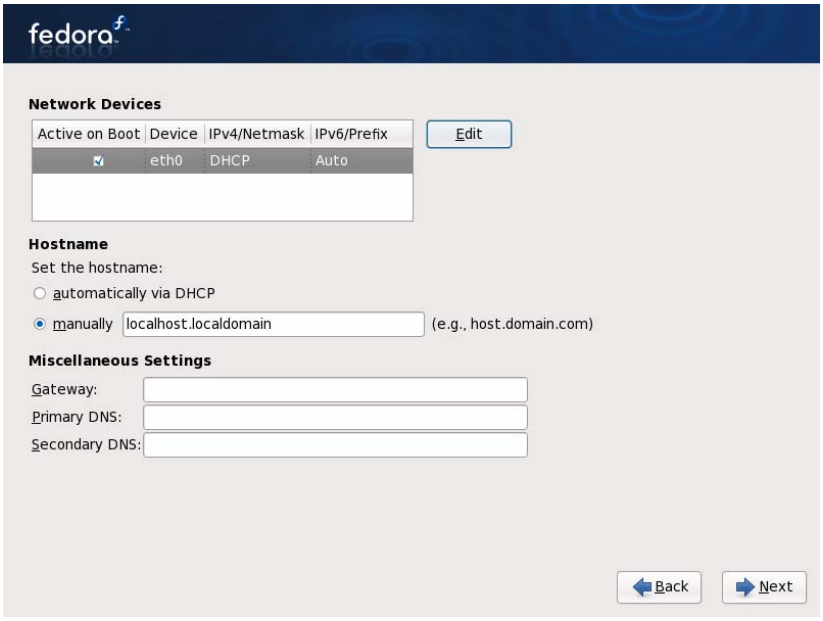
Step4：选择安装过程用什麽语言，这里选择的是英文



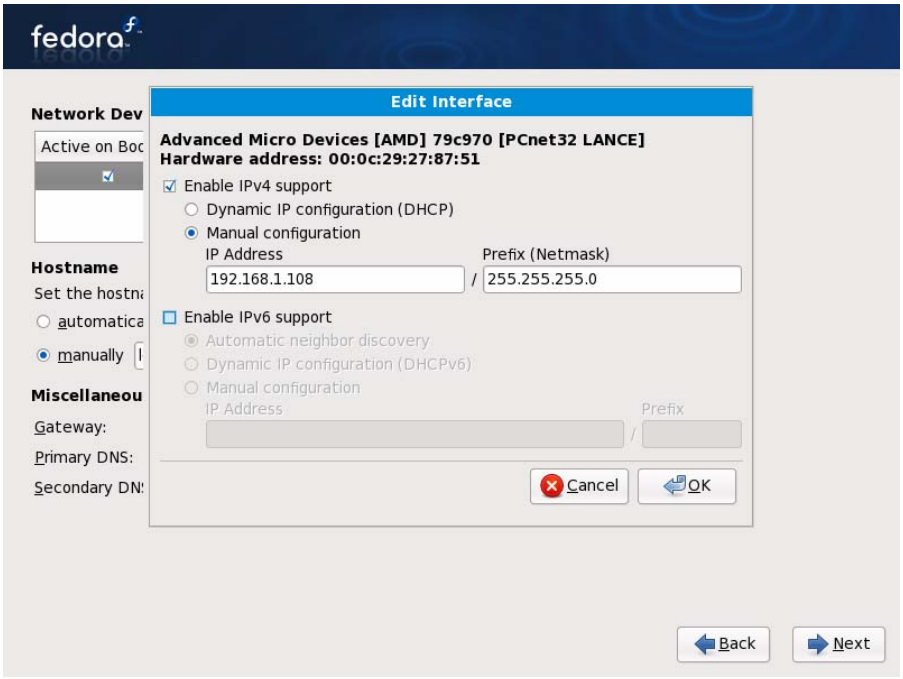
Step5：选键盘，我们一般选美式键盘即可



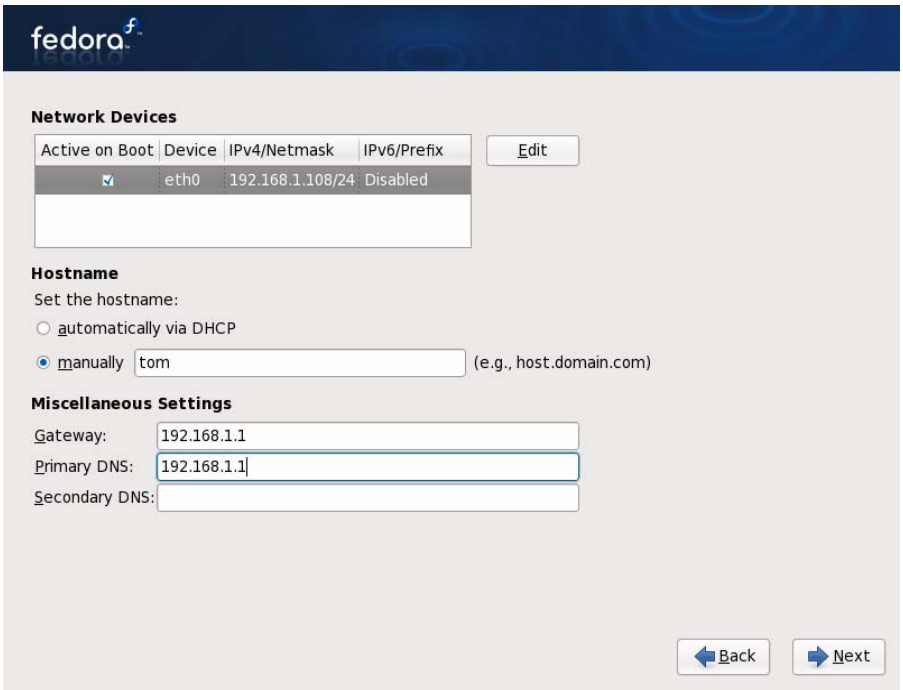
Step6：开始设置网络



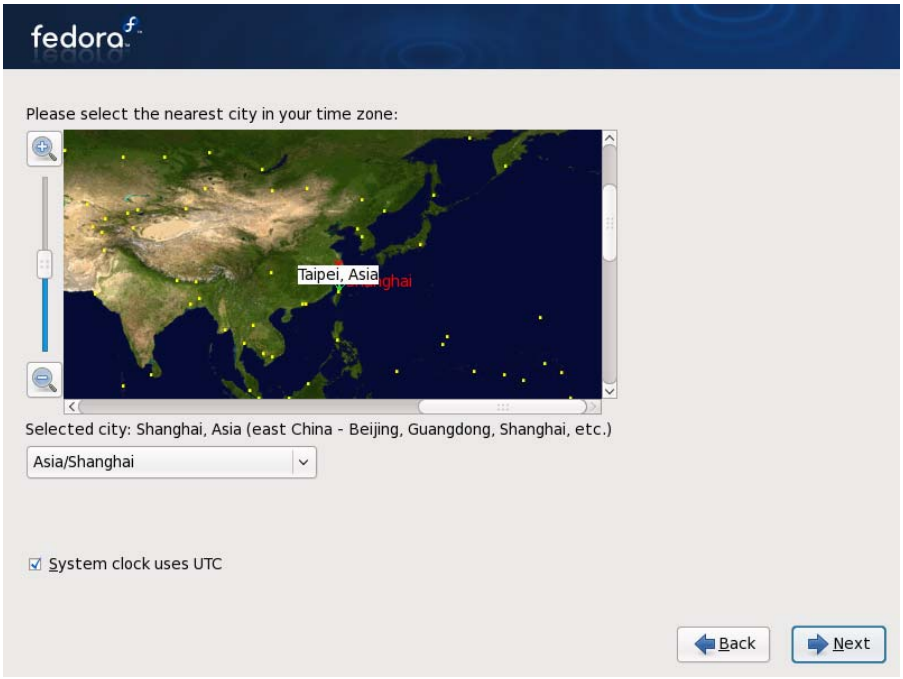
点“Edit”按钮，不要设置为 DHCP，我们一般使用静态的 IP，对照下面进行填写，分别输入 IP 和子网掩码



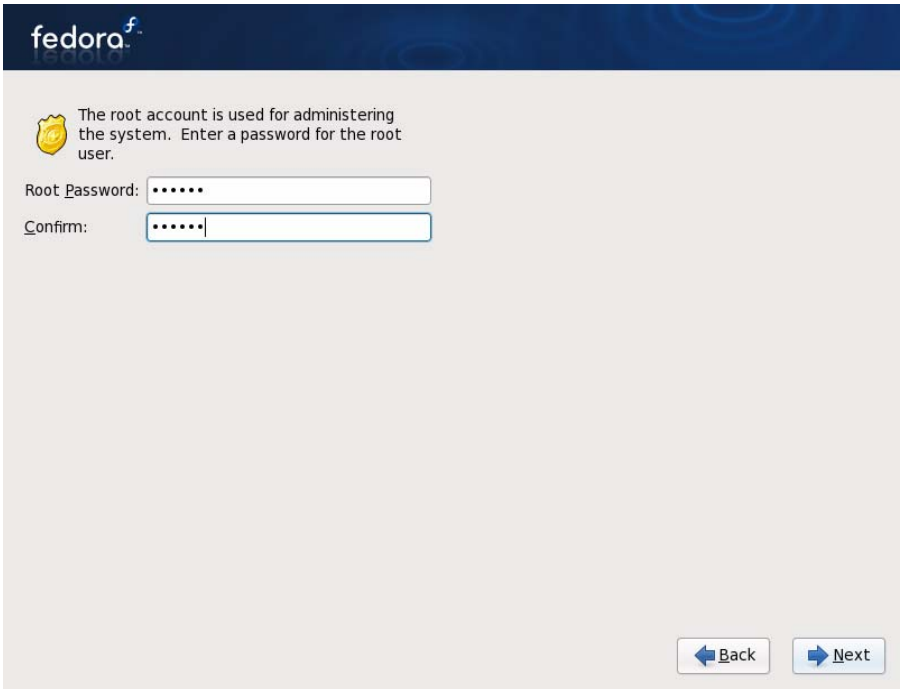
点“OK”返回，开始设置机器名和网关以及 DNS 等



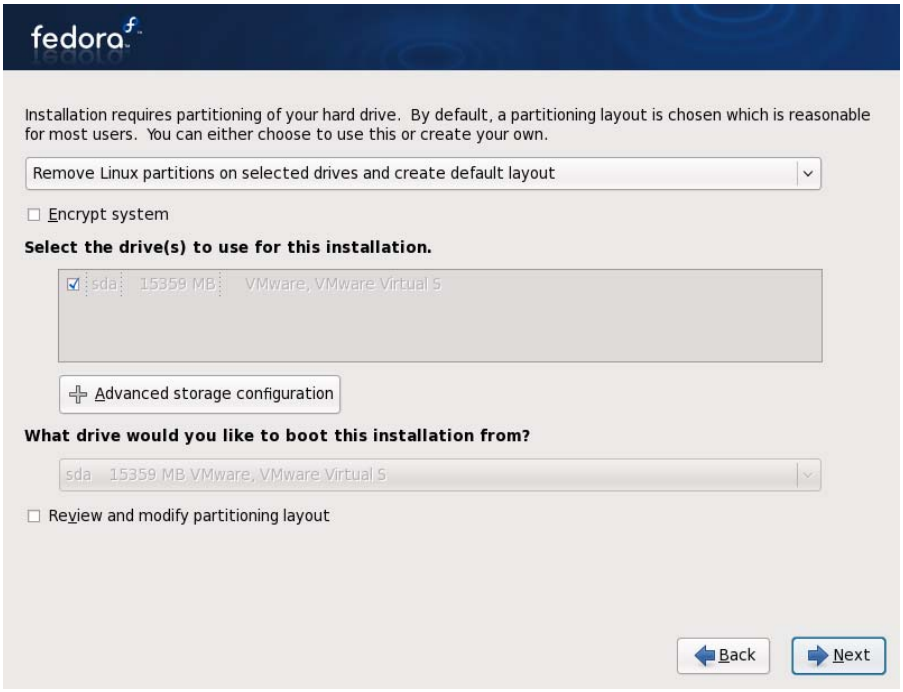
Step7：设置时区，如果你不使用虚拟机安装，“System clock uses UTC”选项可以去掉，如图



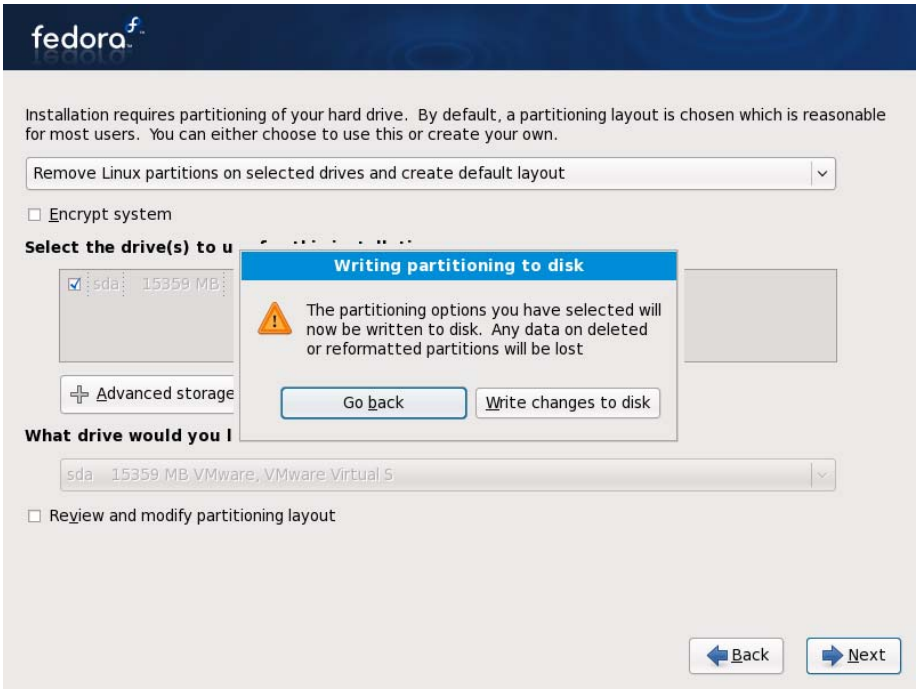
Step8：设置 root 用户密码，必须是 6 位数以上。



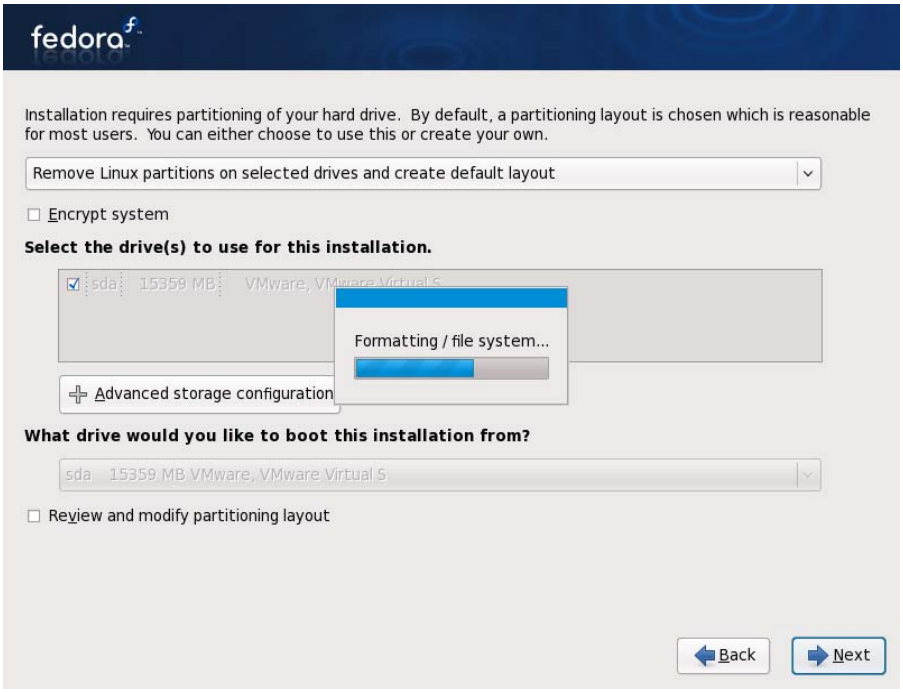
Step9：设置分区，一般选择默认即可，注意要备份好硬盘数据



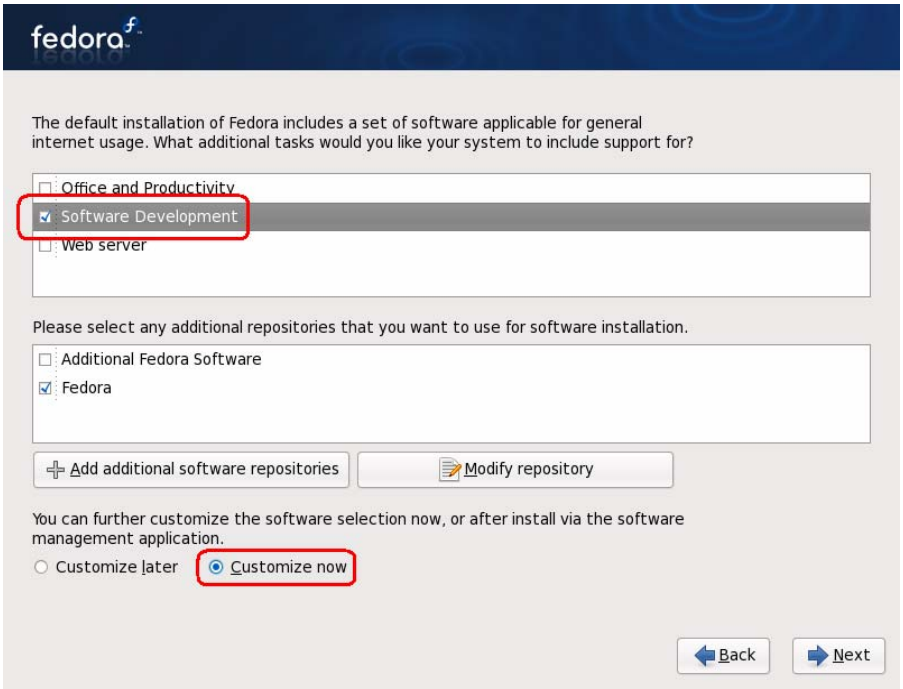
点“Next”会出现警告信息，告诉你继续执行会格式化分区中的所有数据，一般我们在 Vmware 虚拟机中使用，因此可以选“Write changes to disk”，之后开始进行格式化操作。



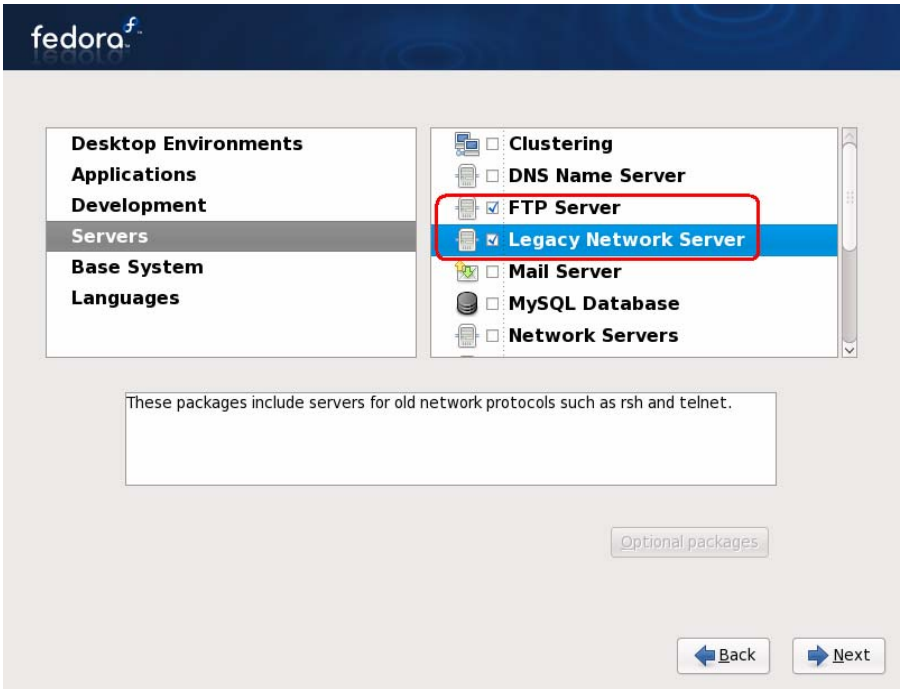
这是格式化的进程图：



Step11：选择安装类型，选择如图，点“Next”开始定制。

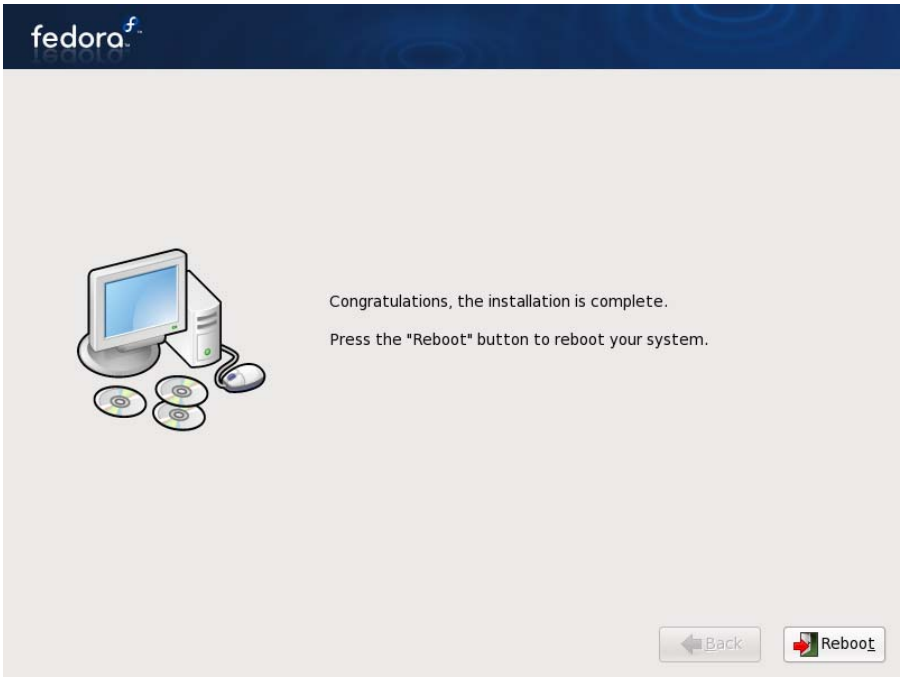


Step12：在 Servers 项中，选择如图



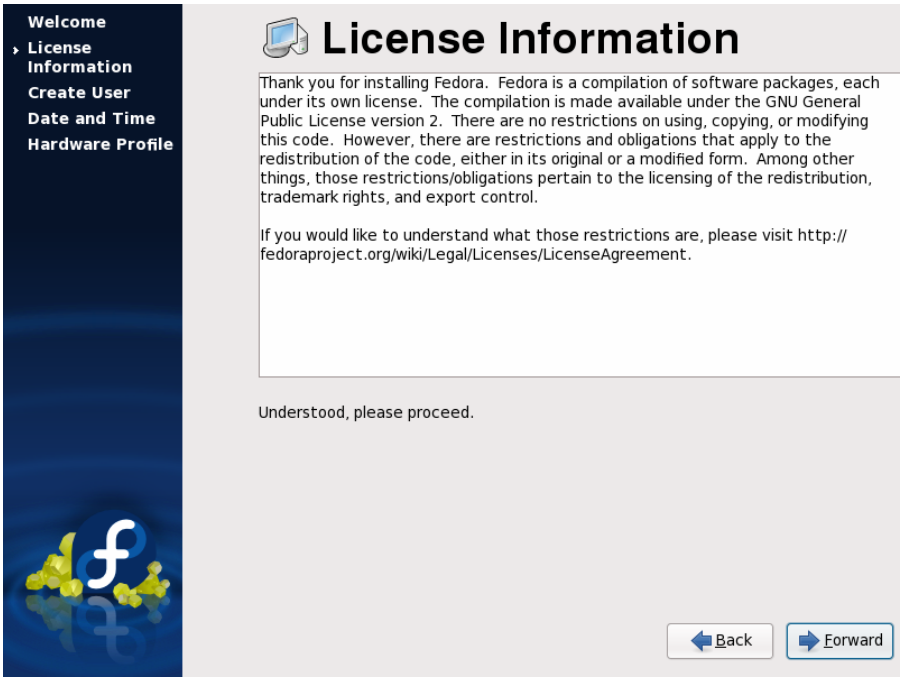
Step13：开始安装系统，此过程时间会比较长，请耐心等待。

Step14：安装完毕，如图

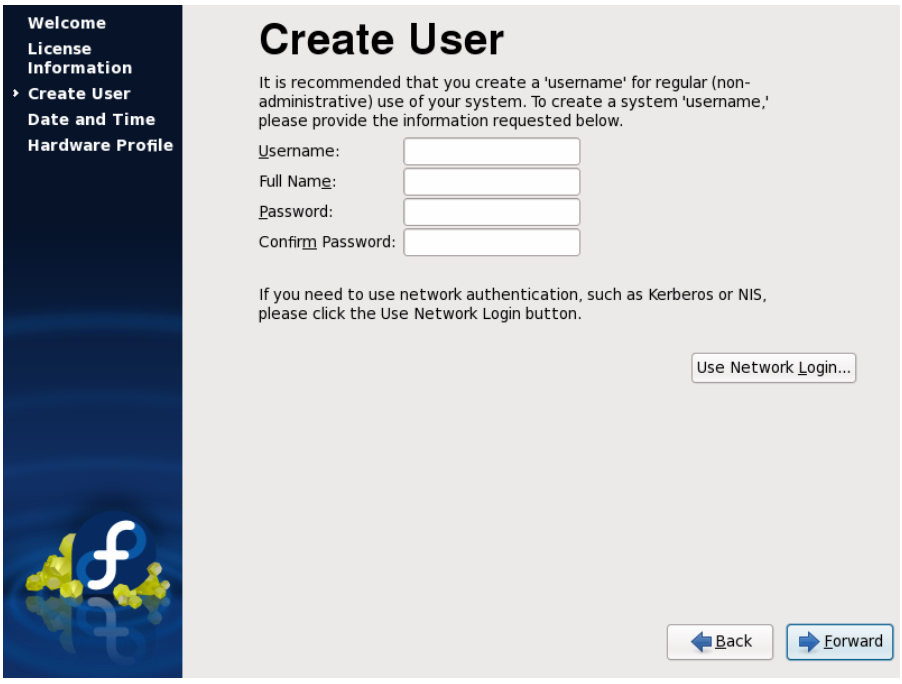


Step15：接上一步，按“Reboot“按钮重启系统，出现第一次使用的界面，如图。

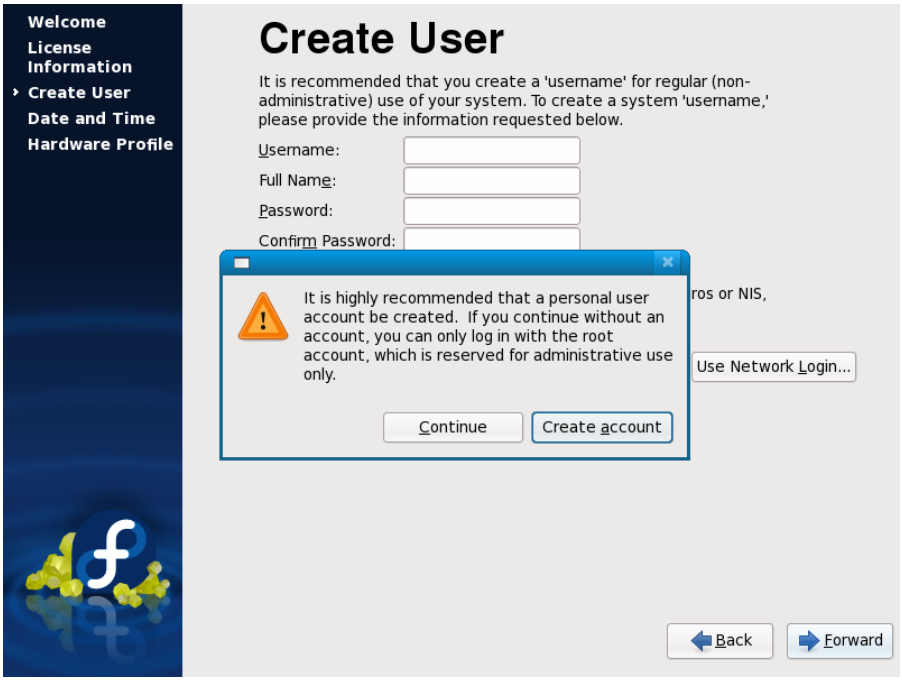
Step16：一些授权信息，不必理会，继续下一步



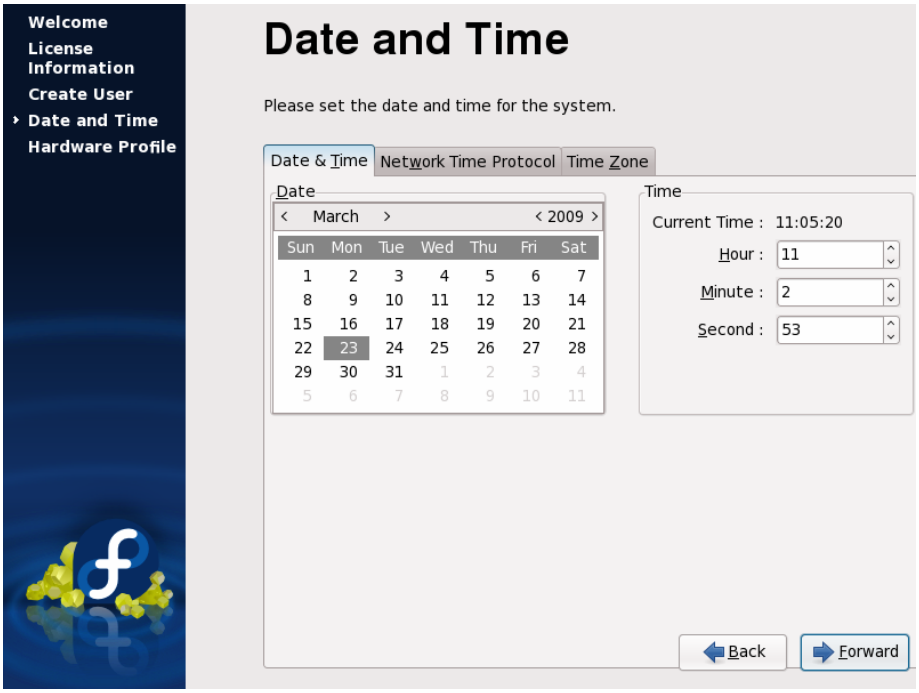
Step17：创建用户，在此我们不需要创建任何新的用户，点“Forward”继续



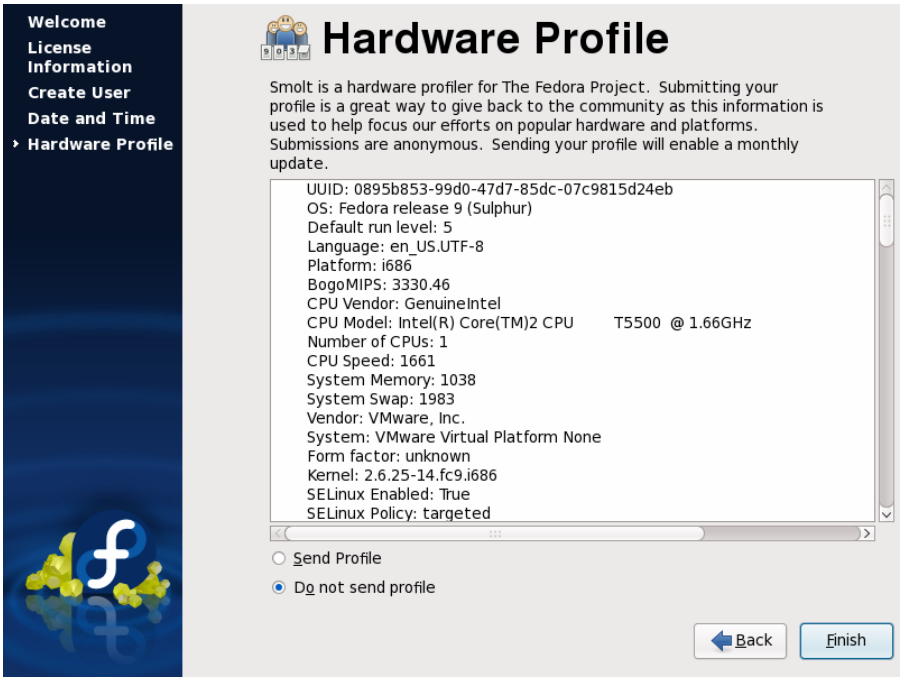
这时会出现提示信息让你确认，点“Continue”继续下一步。



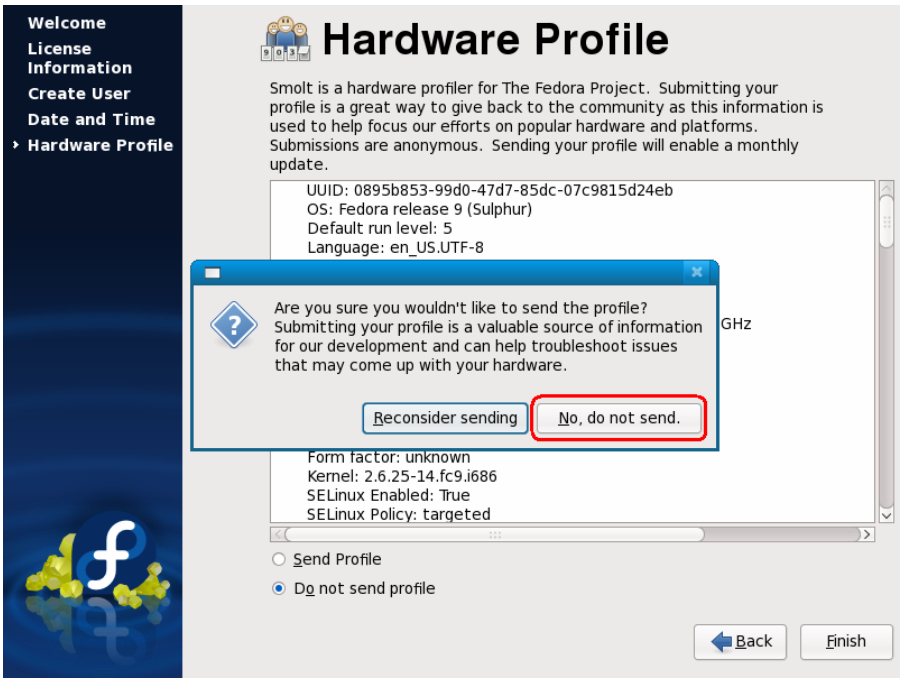
Step18：设置日期和时间，不必理会，继续下一步



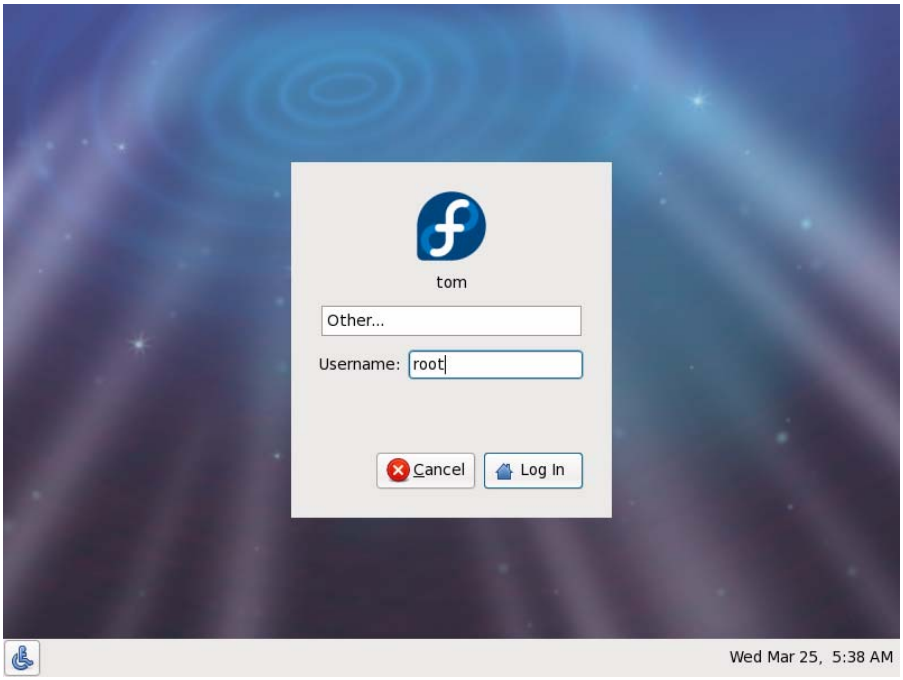
Step19：列出了本机的一些硬件信息，采用缺省设置，点“Finish”



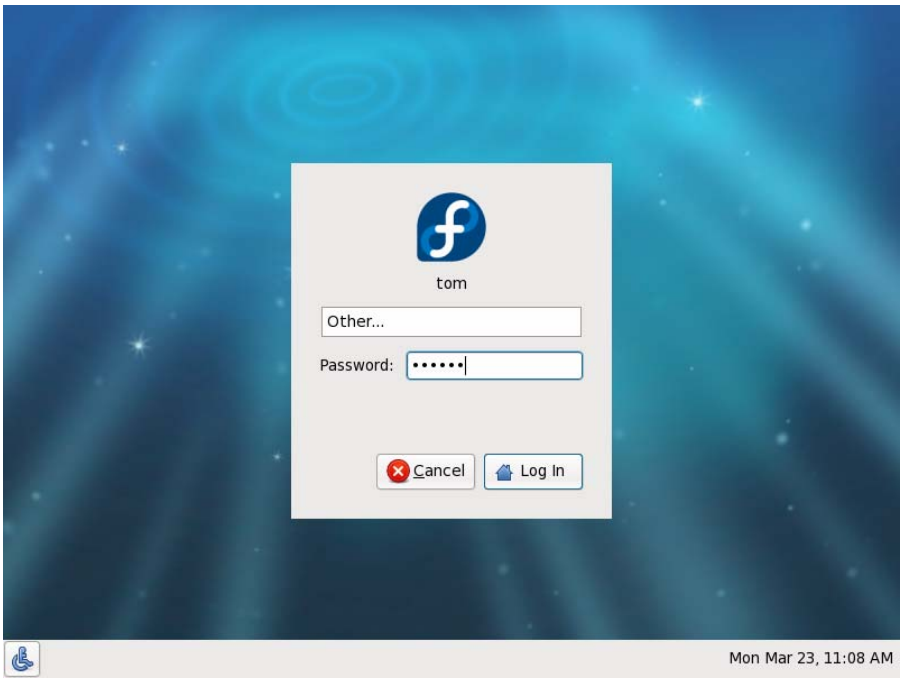
出现提示信息，如图选择，进行下一步



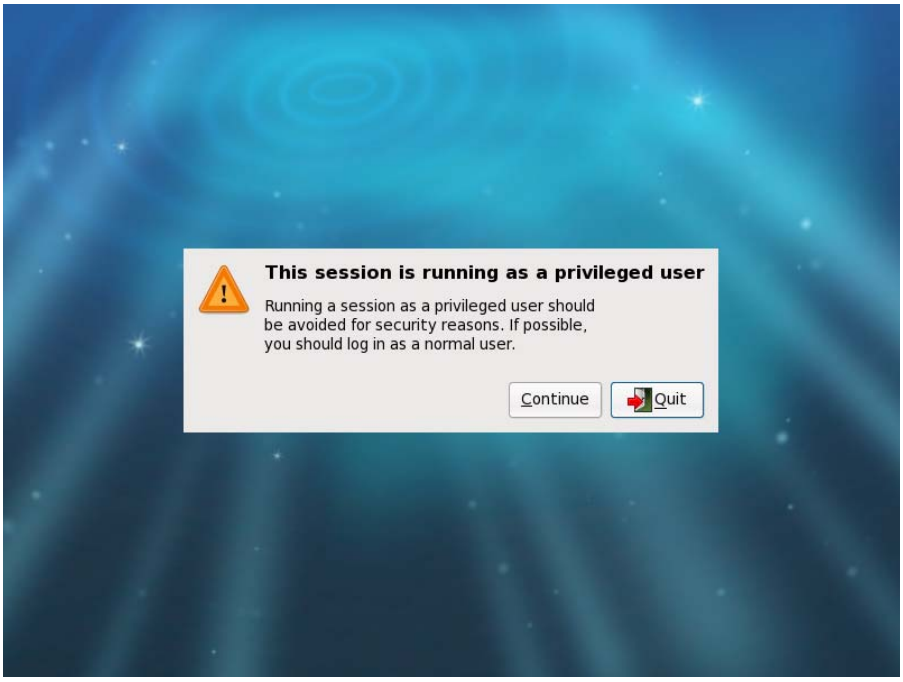
Step20：出现登录界面，我们要以 root 用户进行登录，因此先输入 root



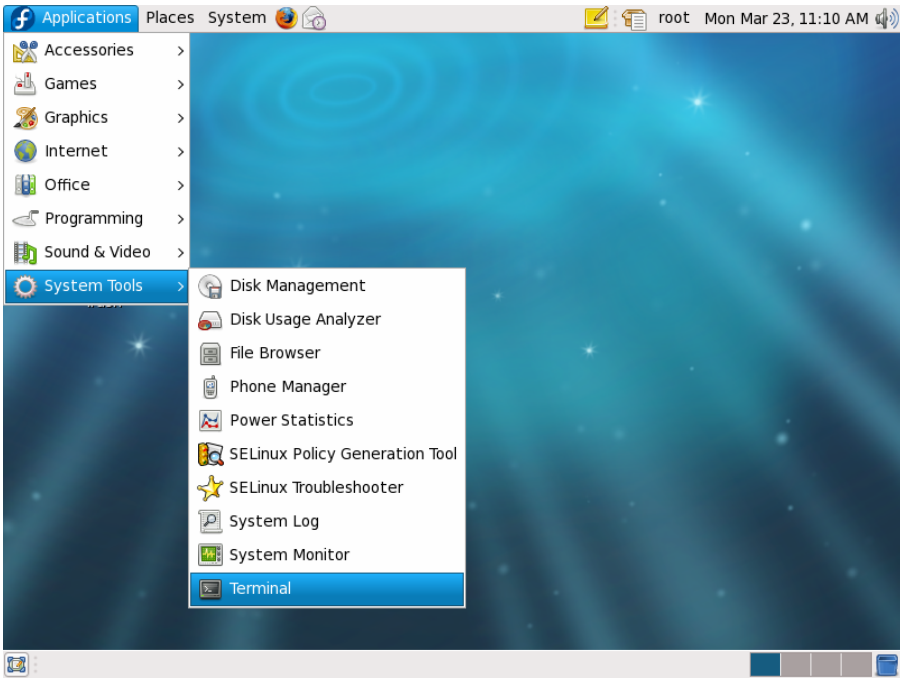
再输入刚才设定的密码



登录后会出现一个提示，以后如果你以 root 用户登录，每次都会出现这个提示，每次均点“Continue”即可。

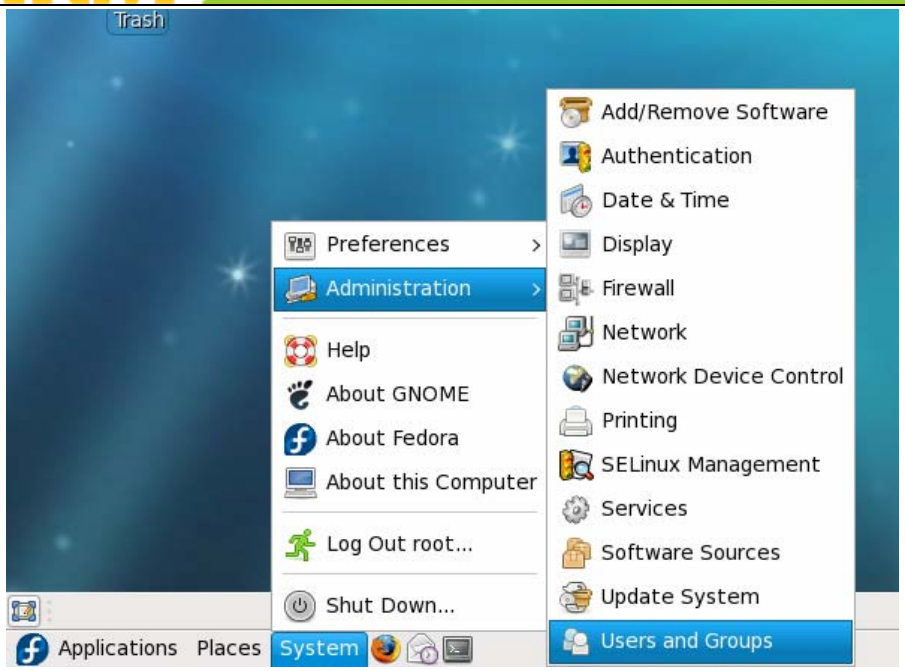


这是登录后的界面，它和 Windows 或者 Ubuntu 是十分类似的。

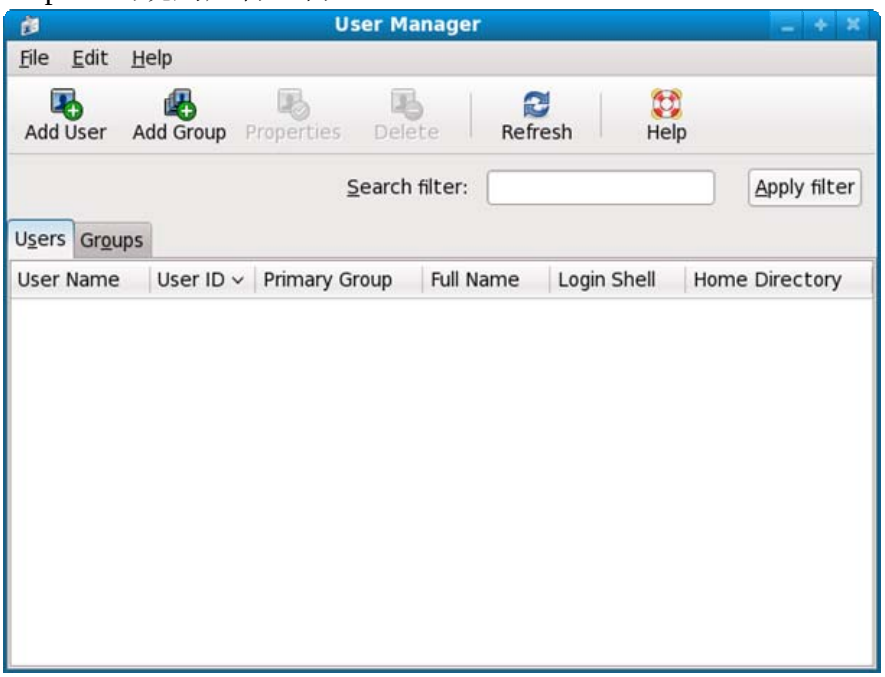


1.3.2 添加新用户

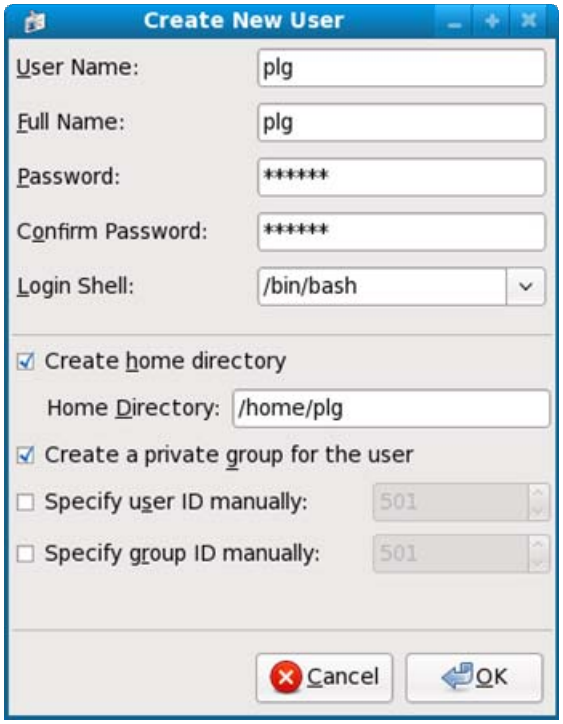
为了方便开发，我们通常创建一个普通权限的用户，步骤如下。
Step1：如图打开用户和组管理器：



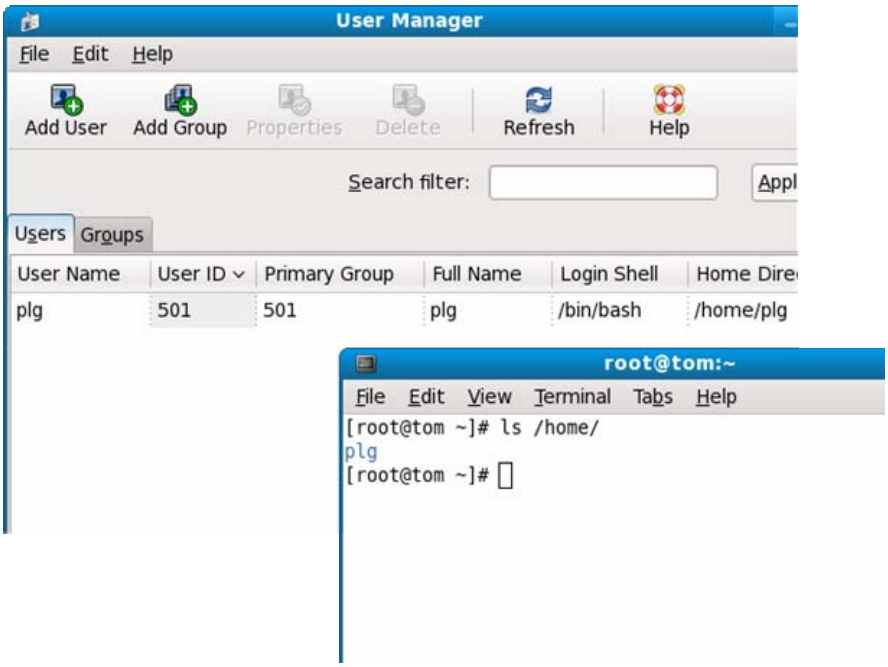
Step2：出现用户管理窗口



Step3：点工具栏的“Add User”按钮，添加新用户，并设置密码：



点“OK”返回，可以看到已经增加了 plg 用户，同时/home 目录下也增加了 plg 用户目录，如图：

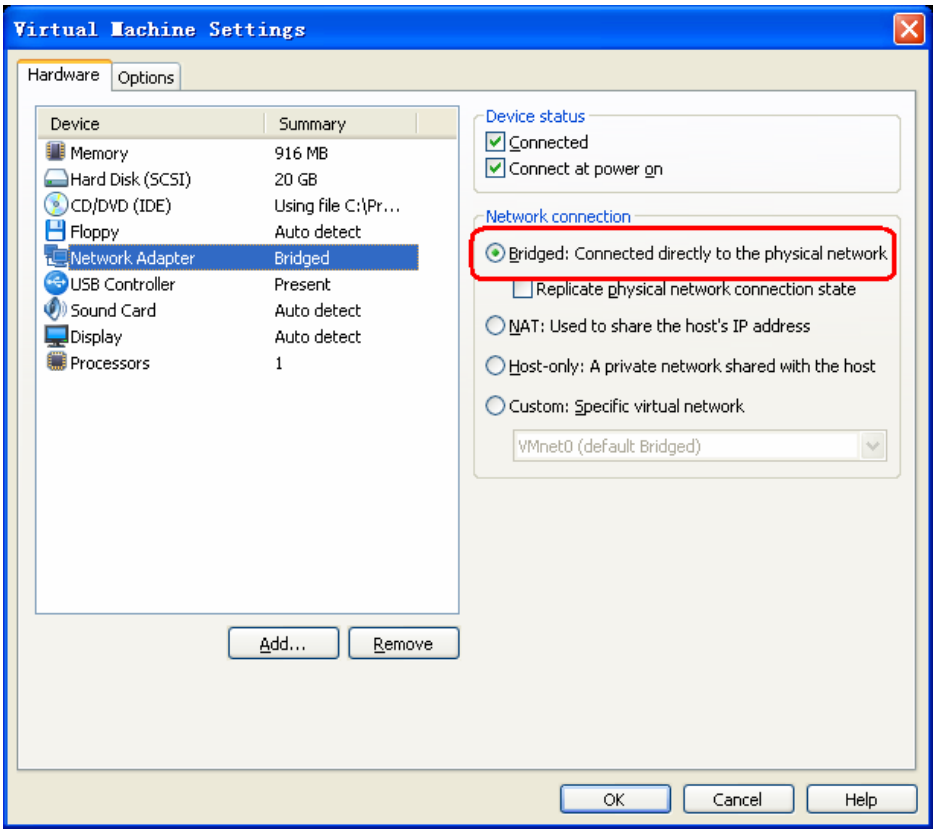


点 Add User 按钮，出现添加新用户窗口，按提示操作就可以了。

1.3.3 访问 Windows 系统中的文件

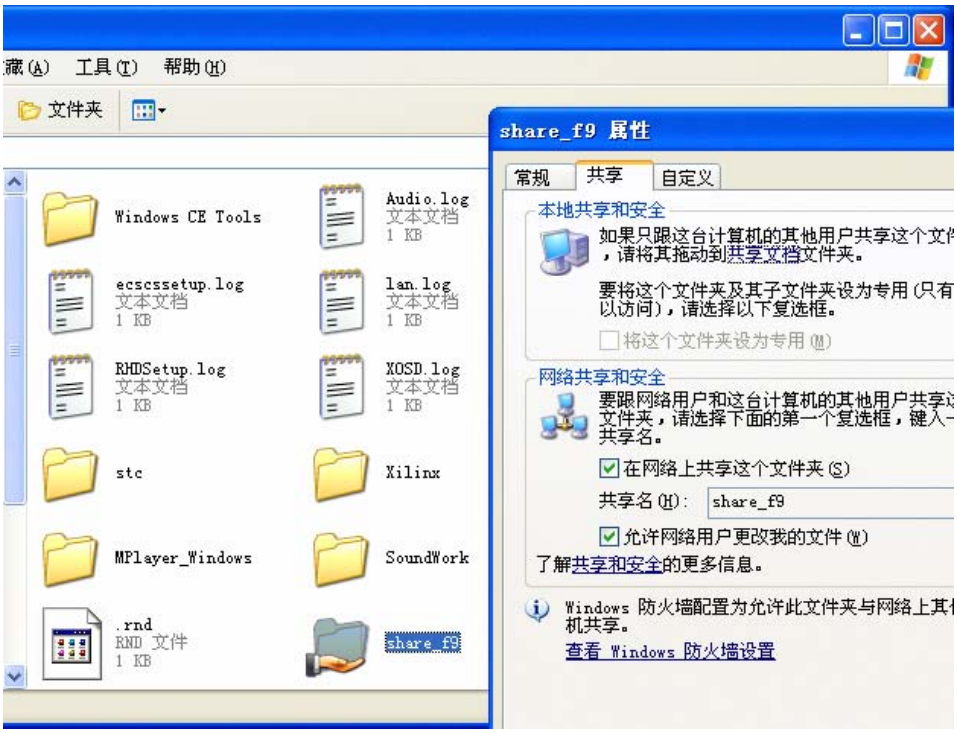
无论你使用的是虚拟机还是真实的 Fedora9 系统，都可以很方便的访问 Windows 中的共享文件，前提是两个系统之间的网络是互通的。

提示：要在虚拟机中使用网络，最简单的方式是设置“Guest”为“Bridges”方式的网络连接，如图：

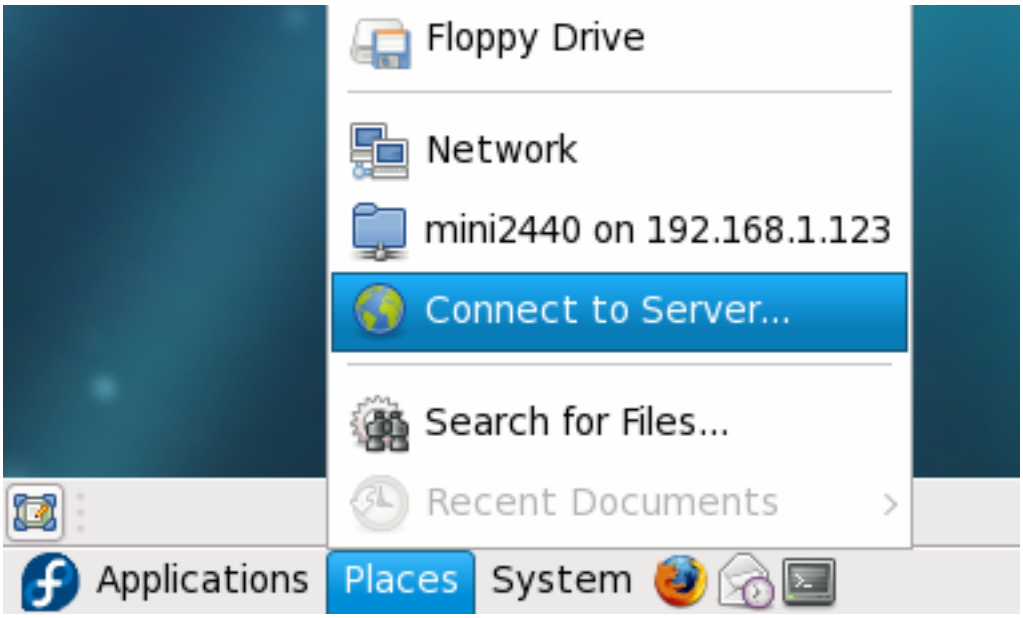


访问 Windows 系统中共享文件的步骤如下：

Step1：在 Windows 中设置共享文件夹“share_f9”(示例)

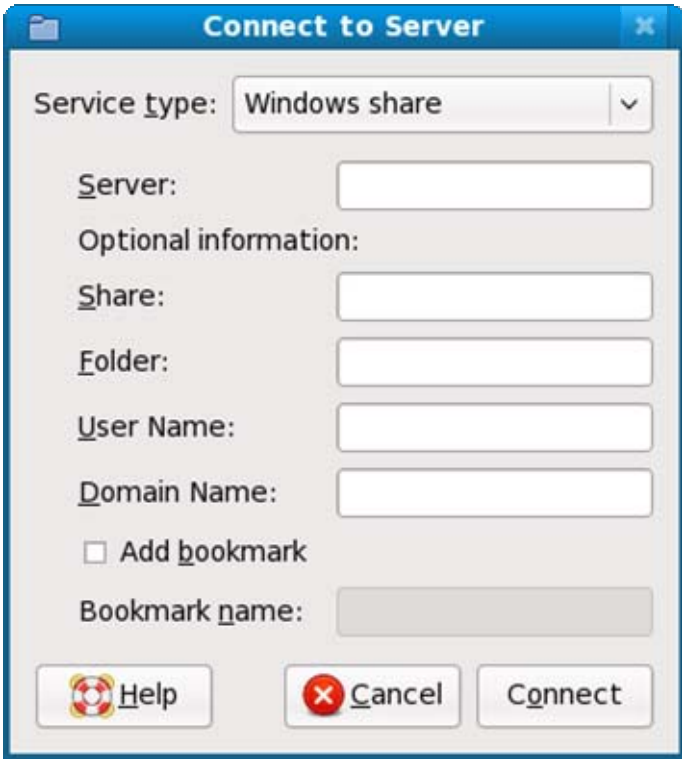


Step2：在 Fedora9 系统中如图操作：

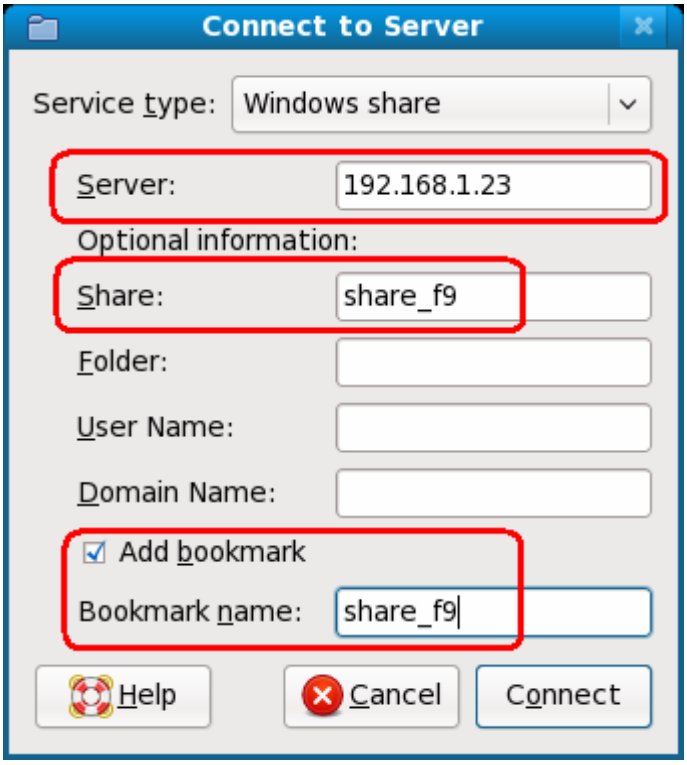


打开如图窗口：

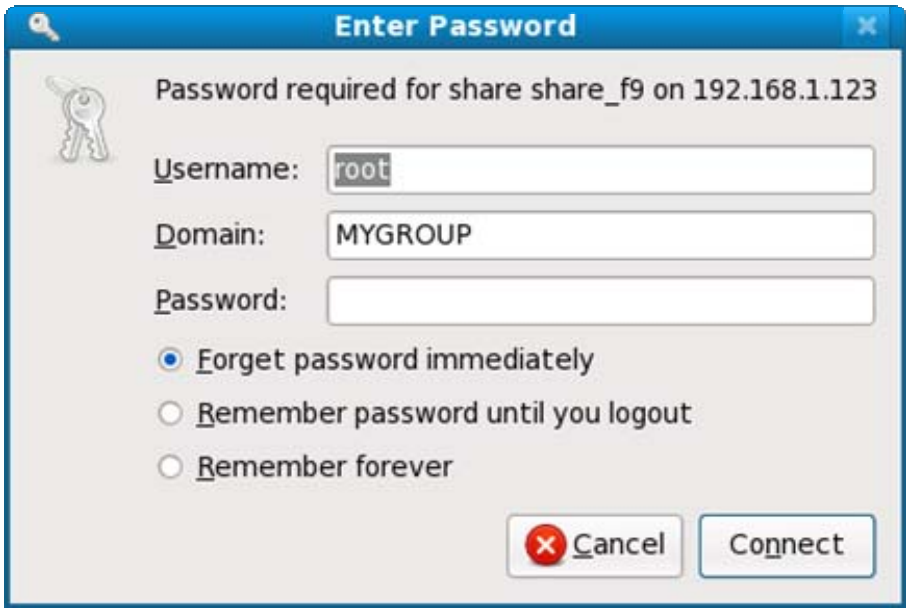
在 Service type 列表中选择 Windows share，如图：



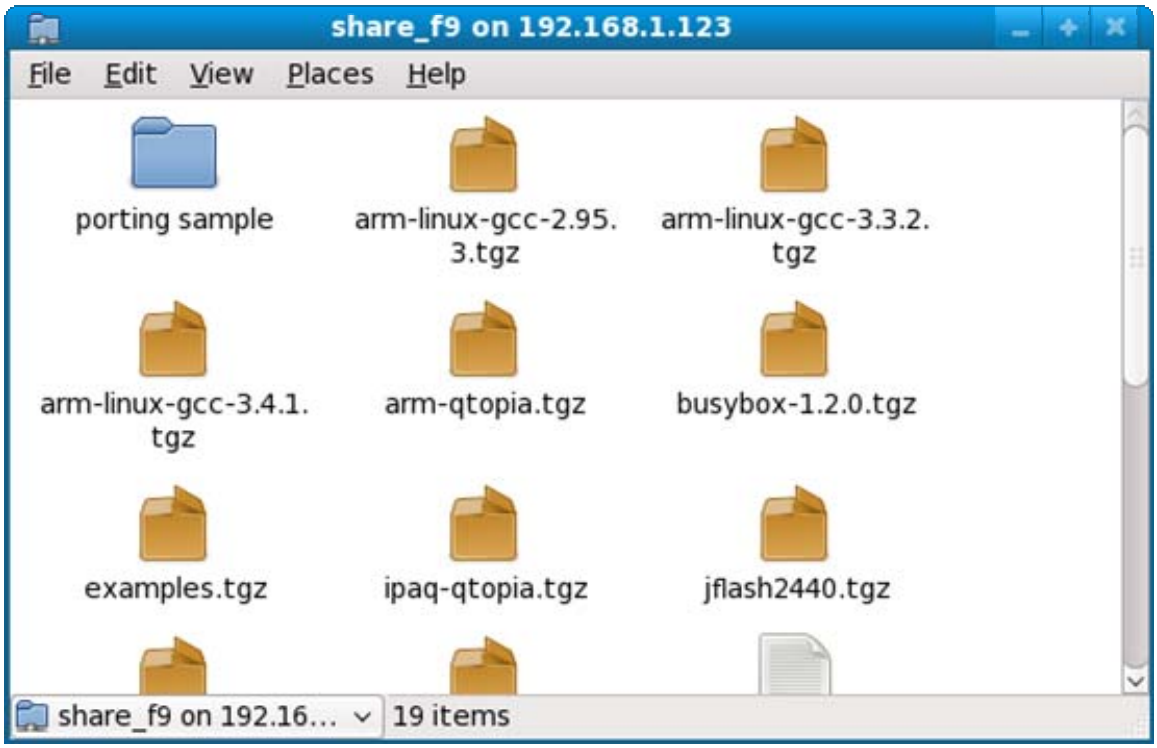
输入所要共享 Windows 主机的 IP 地址和共享文件夹的名字：



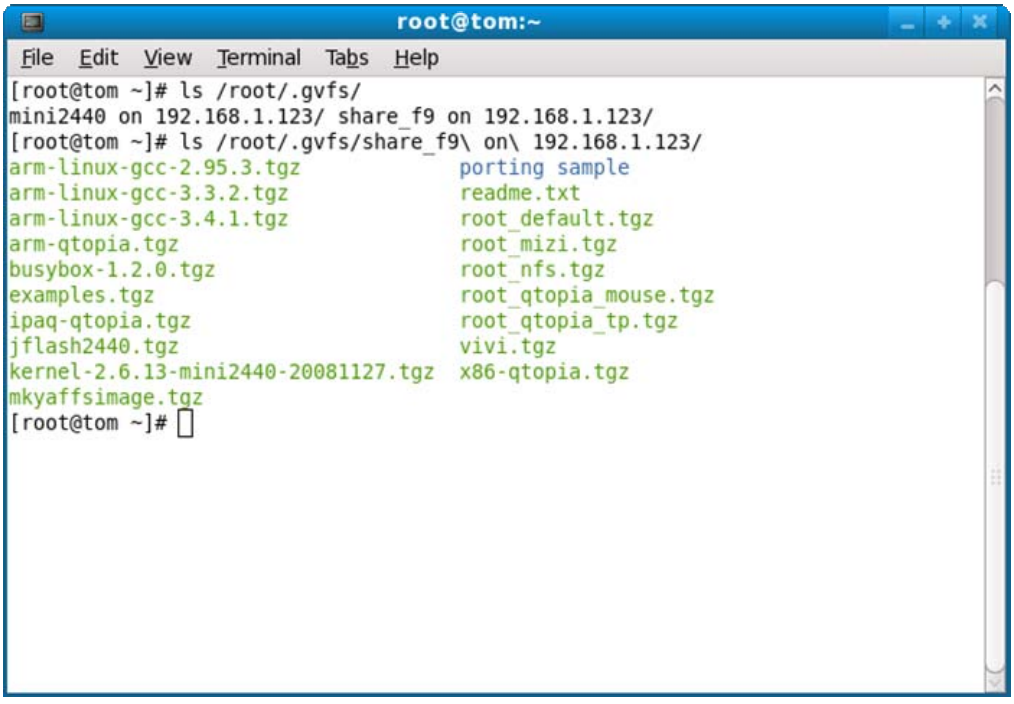
点“Connect”，会出现如下提示窗口：



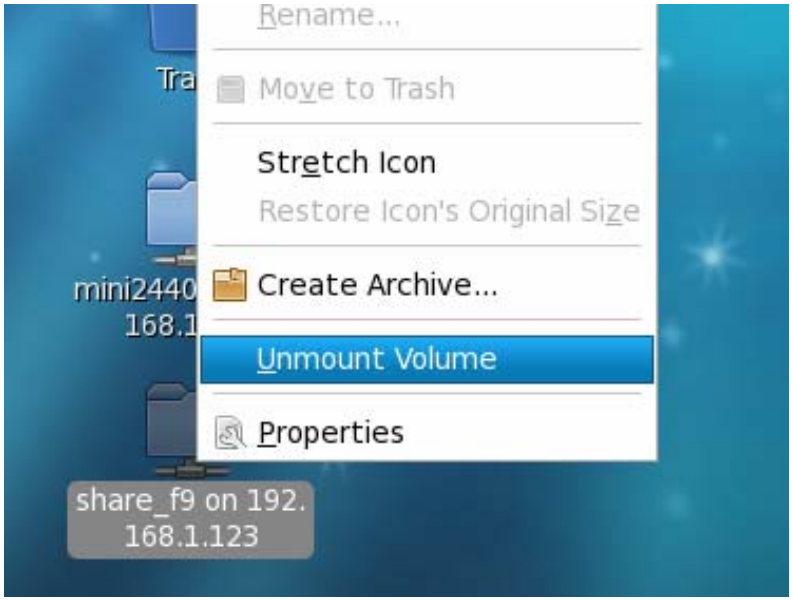
不必理会，直接点“Connect”即可，就可以看到 Windows 共享文件中的内容了，在此你可以像操作其他目录一样来使用它。



如果你想在命令行使用这个目录，可以这样操作：
说明：在控制台下，TAB 键是一个很好使用的小技巧。



要想断开共享目录，只要在桌面的共享文件夹上用右键如图操作就可以了：



1.3.4 配置网络文件系统 NFS 服务

使用本开发板做开发，NFS 服务并不是必须的，因为 NFS 主要是用于通过网络远程共享文件，我们使用常见的 ftp 或者 SD 卡，基本上也可以达到同样的目的。

NFS 服务对于没有接触过 Linux 的人来讲可能比较难以理解，另外，每个人的网络环境也不尽相同，因此设置和使用并没有严格的标准，这就导致初学者比较难以掌握，所以我们并不推荐使用，在此提供的步骤仅供参考；事实上，网络上有很多爱好者根据自己的情况记录了经验总结，你也可以自己搜索看看，关键词是“mini2440 nfs”，它们都是大同小异的，并且可以适用于 6410 系统。

Step1：设置共享目录

以 root 身份登录 Fedora9，在命令行运行：

#gedit /etc/exports

编辑 nfs 服务的配置文件(*注意：第一次打开时该文件是空的*)，添加以下内容：

```
 /opt/FriendlyARM/mini6410/linux/root_qtopia_qt4    *(rw,sync,no_root_squash)
```

其中：

/opt/FriendlyARM/mini6410/linux/root_qtopia_qt4 表示将要共享的的目录，它可以作为开发板的根文件系统通过 nfs 挂接；

* 表示所有的客户机都可以挂接此目录

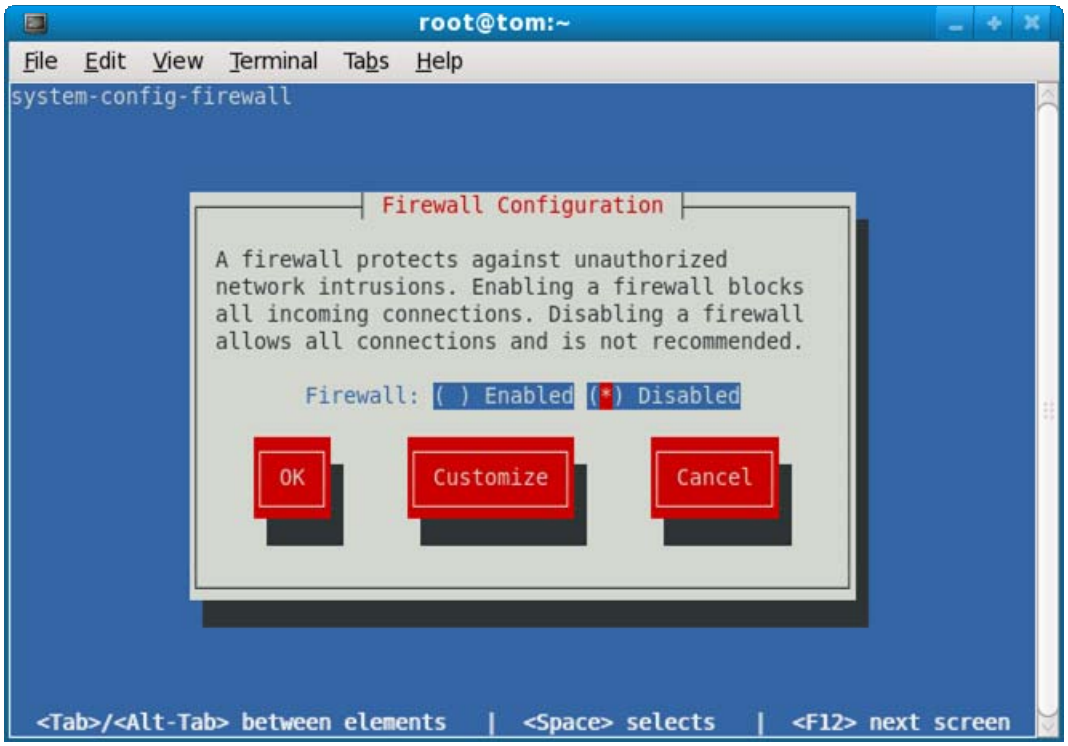
rw 表示挂接此目录的客户机对该目录有读写的权力

no_root_squash 表示允许挂接此目录的客户机享有该主机的 root 身份

说明：实际上该目录目前尚不存在，执行完 4.4 章节的内容之后才会有，在此只是先设置它。

Step2: 启动 NFS 服务

可以通过命令行和图形界面两种方式启动 NFS 服务，我们建立 NFS 服务的目的是通过网络对外提供目录共享服务，但默认安装的 Fedora 系统开启了防火墙，这会导致 NFS 服务无法正常使用。因此先关闭防火墙，在命令行输入“lokkit”命令，打开防火期设置界面：



选择其中(*)Disabled，然后选择“OK”退出，这样就永久的关闭了防火墙。

下面是启动 NFS 服务的方法和步骤：

(1)通过命令启动和停止 nfs 服务

在命令行下运行：

#/etc/init.d/nfs start

这将启动 nfs 服务，可以输入以下命令检验 nfs 该服务是否启动。

mount -t nfs localhost: **/opt/FriendlyARM/mini6410/root_qtopia_qt4** /mnt/

如果没有出现错误信息，您将可以浏览到 **/mnt** 目录中的内容和 **/opt/FriendlyARM/mini6410/root_qtopia_qt4** 是一致的。

使用这个命令可以停止 nfs 服务：

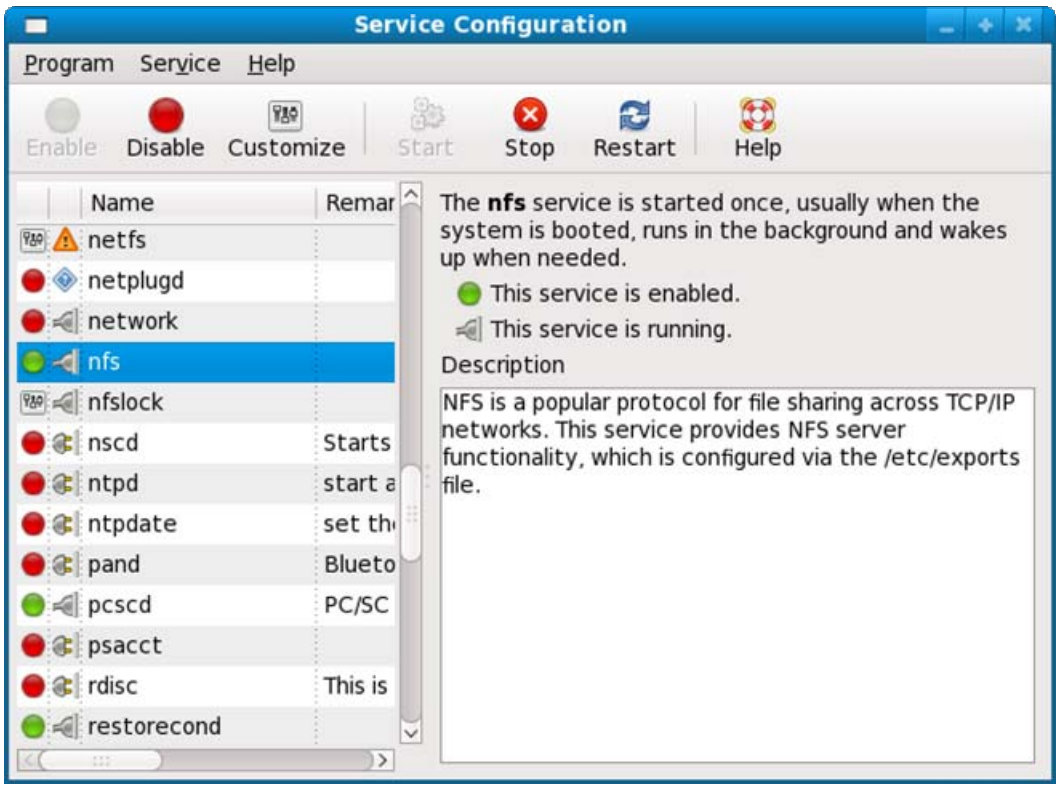
#/etc/init.d/nfs stop

(2)通过图形界面启动 NFS 服务

为了在每次开机时系统都自动启动该服务，可以输入

 # serviceconf

打开系统服务配置窗口，在左侧一栏找到 nfs 服务选项框，并选中它，然后点工具栏的“Enable”启动它，如图。



Step3: 通过 NFS 启动系统

当 NFS 服务设置好并启动后，我们就可以把 NFS 作为根文件系统来启动开发板了。通过使用 NFS 作为根文件系统，开发板的“硬盘”就可以变得很大，因为您使用的是主机的硬盘，这是使用 Linux 作为开发经常使用的方法。

设置开发板为 SDBOOT 启动，注意需要进入菜单模式(见 2“刷机指南”)，连接好电源，串口线，网线；打开串口终端，输入以下命令(不需要加引号)：

```
console=ttySAC0 root=/dev/nfs nfsroot=192.168.1.111:/opt/FriendlyARM/mini6410/root_qtopia_qt4
ip=192.168.1.70:192.168.1.111:192.168.1.111:255.255.255.0:mini6410.arm9.net:eth0:off
```

各参数的含义如下：

nfsroot 是开发主机的 IP 地址，如果你使用了虚拟机，该地址是虚拟机中 Fedora9 的 IP 地址，总之，它是直接提供 NFS 服务的 Linux 系统 IP 地址。

“ip=”后面：

第一项(192.168.1.70)是目标板的临时 IP(注意不要和局域网内其他 IP 冲突)；

第二项(192.168.1.111)是开发主机的 IP；

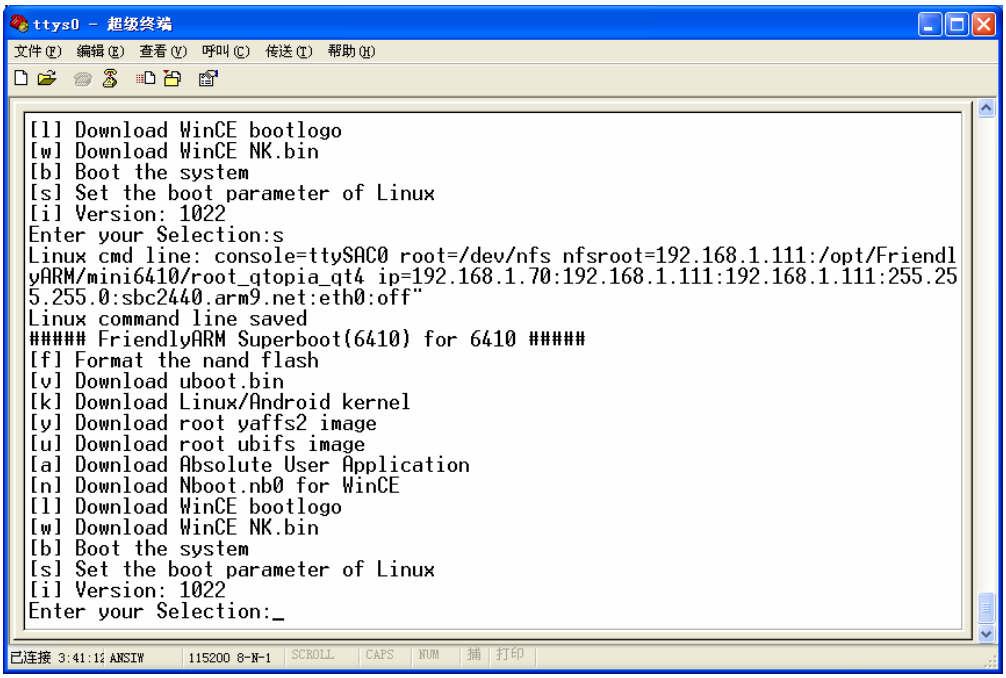
第三项(192.168.1.111)是目标板上网关(GW)的设置；

第四项(255.255.255.0)是子网掩码；

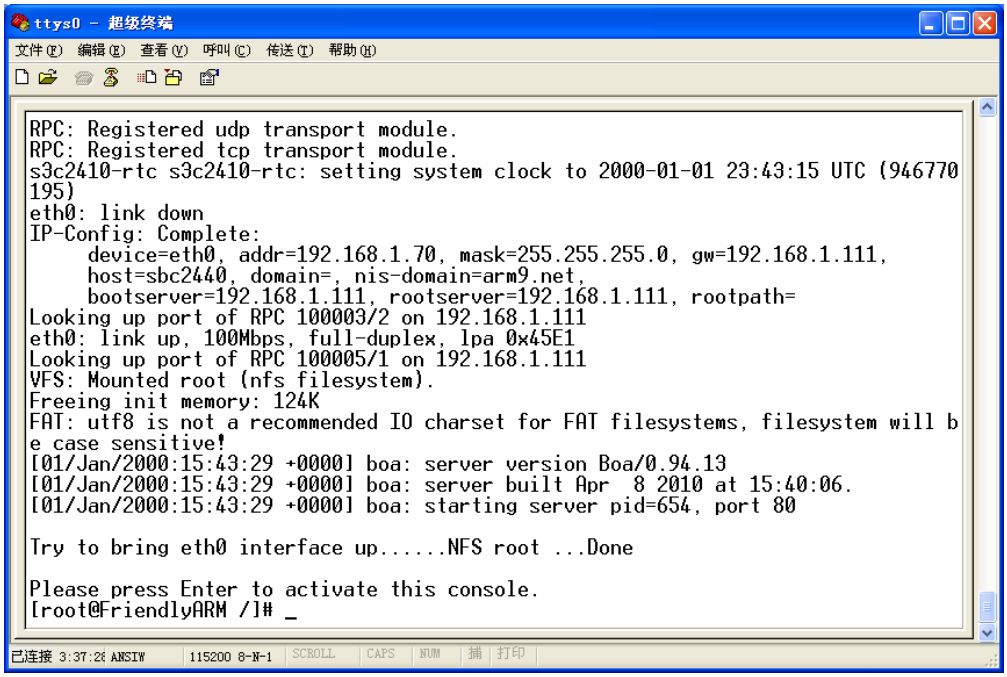
第五项是开发主机的名字(一般无关紧要，可随便填写)

eth0 是网卡设备的名称。

由于该命令比较长，容易输入错误，我们已经把它写入了光盘的 nfs.txt 文件中，这样您直接复制过来就可以了，如图，回车后，该启动参数将被自动保存在 NAND 中。



然后输入 b，按回车就可以通过 nfs 启动系统了。



1.3.5 建立交叉编译环境

在 Linux 平台下，要为开发板编译内核，图形界面 Qtopia/Qt4，bootloader，还有其他一些应用程序，均需要交叉编译工具链，我们使用的是 arm-linux-gcc-4.5.1，它默认采用 armv6

指令集，支持硬浮点运算，下面是安装它的详细步骤。

Step1：将光盘 Linux 目录中的 arm-linux-gcc-4.5.1-v6-vfp-20101103.tgz 复制到 Fedora9 某个目录下如 tmp/，然后进入到该目录，执行解压命令：

#cd /tmp

#tar xvzf arm-linux-gcc-4.5.1-v6-vfp-20101103.tgz　–C /

注意：C 后面有个空格，并且 C 是大写的，它是英文单词“Change”的第一个字母，在此是改变目录的意思。

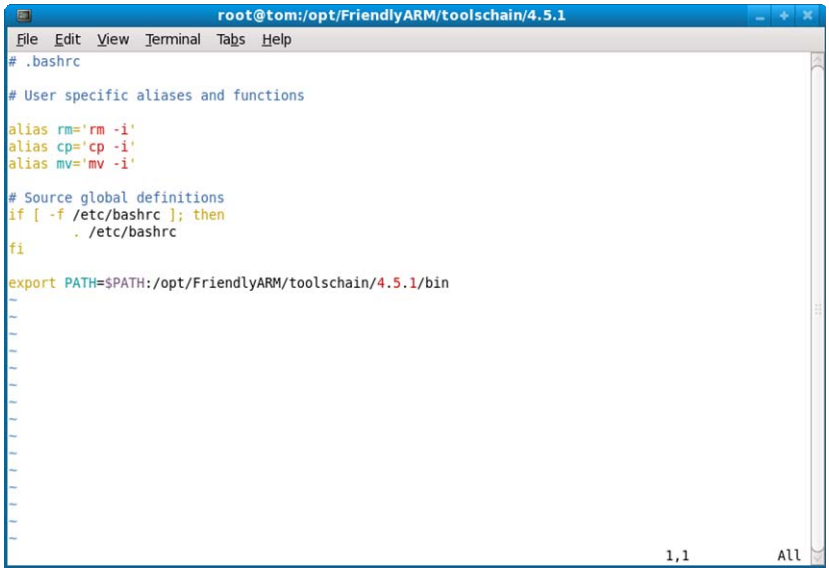
执行该命令，将把 arm-linux-gcc 安装到**/opt/FriendlyARM/toolschain/4.5.1** 目录。

Step2：把编译器路径加入系统环境变量，运行命令

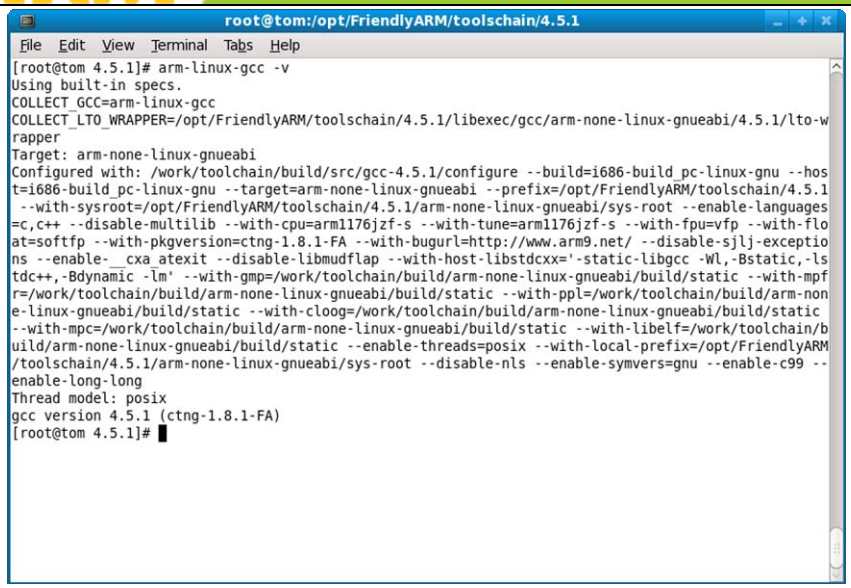
#gedit /root/.bashrc

编辑/root/.bashrc 文件，注意“bashrc”前面有一个“.”，修改最后一行为 **export PATH=$PATH: /opt/FriendlyARM/toolschain/4.5.1/bin**，注意路径一定要写对，否则将不会有效。

如图，保存退出。



重新登录系统(不必重启机器，**开始->logout** 即可)，使以上设置生效，在命令行输入 arm-linux-gcc –v，会出现如下信息，这说明交叉编译环境已经成功安装。



1.4 解压安装源代码及其他工具

本小节将解压安装开发学习过程所用到的全部源代码以及其他一些小工具，这包括：
- Linux 内核源代码
- Qtopia-2.2.0 平台源代码(分为 x86 和 arm 平台两个版本)
- arm-qt-extended-4.4.3 平台源代码(也就是 Qtopia4，分为 x86 和 arm 两个版本)
- QtE-4.7.0 平台源代码(arm 版本)
- busybox-1.17 源代码
- Linux 编程示例源代码
- U-boot 源代码
- 目标文件系统目录
- 目标文件系统映象制作工具(包括 yaffs2 和 UBIFS)
- 图形界面的 Linux logo 制作工具 **logomaker**

注意：所有的源代码和工具都是通过解压方式安装的，所有的源代码均使用统一的编译器 arm-linux-gcc-4.4.1 编译

下面是详细的解压安装过程，并有简要的介绍。

1.4.1 解压安装源代码

首先创建工作目录/opt/FriendlyARM/mini6410/linux
在命令行执行
#mkdir –p /opt/FriendlyARM/mini6410/linux
后面步骤的所有源代码都会解压安装到此目录中：

(1)准备好 Linux 源代码包

在 Fedora9 系统中/tmp 目录中创建一个临时目录**/tmp/linux**

#mkdir /tmp/linux

把光盘中 linux 目录中的所有文件都复制到**/tmp/linux** 目录中

说明：这样做是为了统一下面的操作步骤，其实你可以使用其他目录，也可以直接从光盘解压安装。

(2)解压安装 U-boot 源代码

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ u-boot-mini6410-20101106.tar.gz

将创建生成 u-boot-mini6410 目录，里面包含了完整的 U-boot 源代码

说明：20101106 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。

(3)解压安装 Linux 内核源代码

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ linux-2.6.36-20101115.tar.gz

将创建生成 linux-2.6.36 目录，里面包含了完整的内核源代码

说明：20101115 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。

(4)解压安装目标文件系统

执行以下命令：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ rootfs_qtopia_qt4-20101120.tgz

将创建生成 rootfs_qtopia_qt4 目录

说明：20101120 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。

(5)解压安装嵌入式图形系统 qtopia 源代码

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ x86-qtopia-20100420.tar.gz

#tar xvzf /tmp/linux/ arm-qtopia-20101105.tar.gz

将创建 x86-qtopia 和 arm-qtopia 两个目录，并内含相应的全部源代码。

说明：x86-qtopia 和 arm-qtopia 后面或许会有日期尾缀，它是我们的发行或更新日期标志，请以光盘中实际日期尾缀为准。源代码包中也包含了嵌入式浏览器 konquor 的源代码。

另外，为了方便用户学习开发使用，此源代码包相比 Qt 公司的原始版本已经打过补丁，并做了诸多改进，它们都是源代码方式，我们不再一一赘述，感兴趣者可自行比较。

(6)解压安装嵌入式图形系统 qt-extended-4.4.3 源代码

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ x86-qt-extended-4.4.3-20101003.tgz

#tar xvzf /tmp/linux/ arm-qt-extended-4.4.3-20101105.tgz

将创建 x86-qt-extended-4.4.3 和 arm-qt-extended-4.4.3 两个目录，并内含相应的全部源代码。

说明：x86-qt-extended-4.4.3 和 arm-qt-extended-4.4.3 后面或许会有日期尾缀，它是我们的发行或更新日期标志，请以光盘中实际日期尾缀为准。

另外，为了方便用户学习开发使用，此源代码包相比 Qt 公司的原始版本已经打过补丁，并做了一些改进，它们都是源代码方式，我们不再一一赘述，感兴趣者可自行比较。

(7)解压安装 QtE-4.7.0 源代码

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/x86-qte-4.6.1-20100516.tar.gz

#tar xvzf /tmp/linux/ arm-qte-4.7.0-20101105.tar.gz

将创建 x86-qte-4.6.1 和 arm-qte-4.7.0 两个目录，并内含相应的全部源代码。

说明：x86-qte 和 arm-qte 压缩包后面或许会有日期尾缀，它是我们的发行或更新日期，请以光盘中实际日期尾缀为准。其中 x86-qte-4.6.1 主要是创建 Creator 开发平台，版本稍微低一些不会影响开发。

(8)解压安装 busybox 源代码

Busybox是一个轻型的linux命令工具集，在此使用的是busybox-1.13.3 版本。用户可以从其官方网站下载最新版本(http://www.busybox.net)。

在工作目录/opt/FriendlyARM/mini6410/linux 中执行：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ busybox-1.17.2-20101120.tgz

将创建 busybox-1.17.2 目录，内含相应版本的全部源代码。

说明：为了方便用户编译使用，我们做了一个缺省的配置文件 fa.config。

(9)解压安装 Linux 示例程序

执行以下命令：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ examples-mini6410-20101110.tgz

将创建 examples 目录，并包含初学 linux 编程代码示例。

说明：20101110 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。examples 目录中的代码均为友善之臂自主开发，并全部以源代码方式提供，它们都是一些基于命令行的小程序。

1.4.2 解压创建目标文件系统

根据触摸屏的连接配置方式，为了方便用户使用，我们制作了 2 个目标文件系统压缩包：

- rootfs_qtopia_qt4-20101120.tgz
- rootfs_qtopia_qt4-s-20101120.tgz

其中：带“-s”的表示适合于采用专业串口触摸屏控制器的 LCD 套餐，它适合于大尺寸的四线电阻触摸屏，会达到更好的效果；不带“-s”的表示采用了 ARM 本身触摸屏控制器，或一线触摸屏(第一次运行时会自动识别)。它们的唯一不同之处在于“/etc/friendlyarm-ts-input.conf”配置文件的定义。

执行以下命令：

#cd /opt/FriendlyARM/mini6410/linux

#tar xvzf /tmp/linux/ rootfs_qtopia_qt4-20101120.tgz

tar xvzf /tmp/linux/ rootfs_qtopia_qt4-s-20101120.tgz

将创建 rootfs_qtopia_qt4 和 rootfs_qtopia_qt4-s 两个目录，该目录和目标板上使用的文件系统内容是完全一致的。

说明：20101120 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准，该文件系统包含了前面你所看到的 qtopia-2.2.0，Qtopia4 和 QtE-4.7.0 测试软件等，busybox，还有常用的命令行工具等，和之前的相比，它具有如下特性：

- 自动识别 NFS 启动或本地启动
- 自动识别所接的输出显示模块是否接了触摸屏，以判断在第一次开机使用时是否要进行校正。如果没有连接，会自动进入系统，使用鼠标即可；否则会先校正触摸屏。
- 自动识别普通或者高速 SD 卡(最大可支持 32G)和优盘
- 自动检测 USB 鼠标或触摸屏
- 支持 USB 鼠标和触摸屏共存(自 Linux-2.6.36 开始支持)

1.4.3 解压安装文件系统映像工具

要把目标文件系统全部写入开发板中，一般还需要先把目标文件系统目录制作成单个的映像文件以便烧写或者复制，Linux 内核启动时，一般会根据命令行参数挂在不同格式的系统，比如 yaffs2, ubifs, ext2, nfs 等。它们一般都是命令行方式的小程序。

针对 64M 或 128M/256M/512M/1GB 的 mini2440/mcro2440，分别有 2 套制作工具：mkyaffs2image 和 mkyaffs2image-128M。其中 mkyaffs2image 是制作适用于 64M 版本文件系统映象的工具，它沿用了以前的名字；mkyaffs2image-128M 是制作适用于 128M/256M/512M/1GB 版本文件系统映象的工具，为了便于区分，我们把它命名为此。

对于 6410 系统而言，我们没有制作 64M 版本，因此可以沿用 2440 系统所用的 **mkyaffs2image-128M**

UBIFS 是近两年流行的另一种针对嵌入式系统的存储器文件系统格式， UBIFS 官方使用的工具比较复杂，并且参数和步骤都很多，为了方便使用，我们也专门设计了 **mkubimage**

另外还有 **mkext3image**，它用来把目标文件系统制作成单个的 EXT3 映像文件，这样就可以在普通的 FAT32/FAT 格式的 SD 卡中安装使用各种类 Linux 系统了，只需要相应的单个系统映像文件复制到 SD 卡中就可以，这不再需要很复杂的步骤。

我们把这些小工具统称为 mktools，下面是它的安装步骤：

执行以下命令：

#tar xvzf /tmp/linux/mktools.tar.gz –C /

将会在/usr/sbin 目录下创建生成 mkyaffs2image, mkyaffs2image-128M, mkubimage, mkext3image

注意：C 是大写的，C 后面有个空格，C 是改变解压安装目录的意思

说明：如果你以前安装过 mini2440 使用的 mkyaffs2image 系列工具，它们将会被覆盖，请不必担心，它们功能都是相同的。

1.4.4 解压安装 LogoMaker

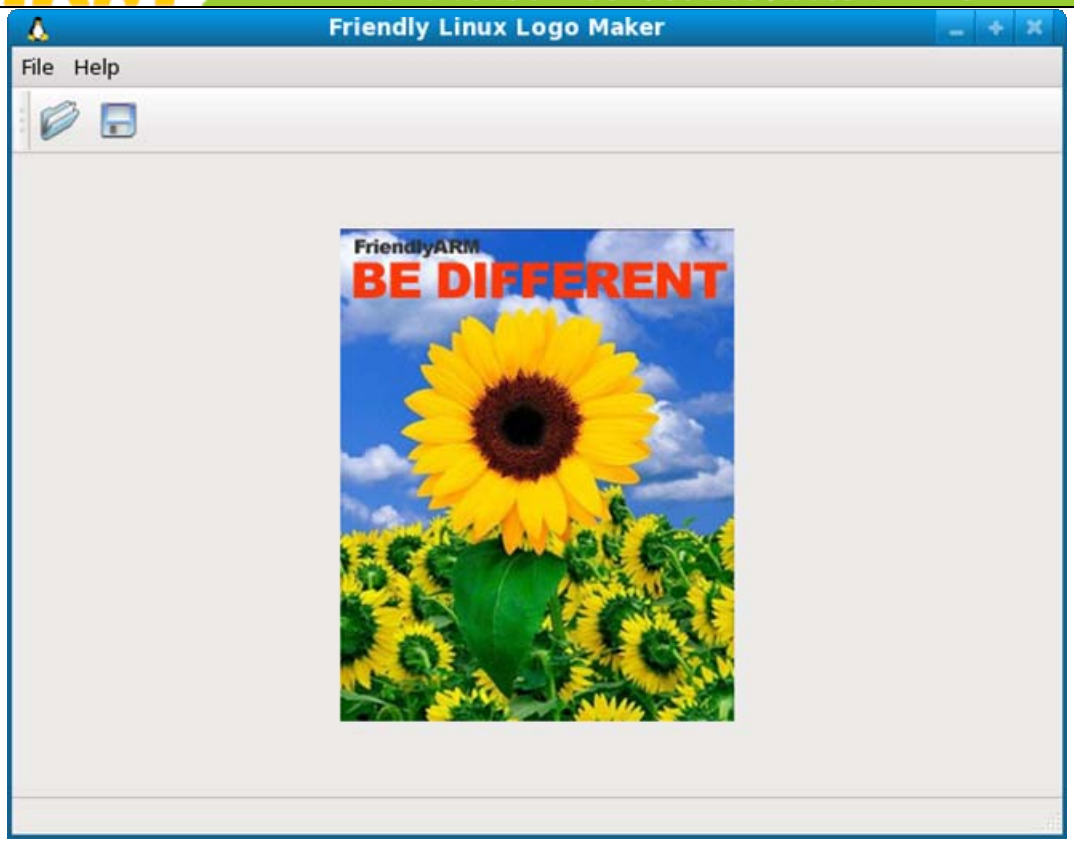
LogoMaker 是友善之臂开发的一个 Linux Logo 简易制作工具，网上有很多资料介绍如何使用命令行的工具把 bmp, jpg, png 等格式的图片转换为 Linux Logo 文件，在此我们设计了一个图形化的版本，它是基于 Fedora9 开发。

执行以下命令：

#tar xvzf /tmp/linux/logomaker.tgz –C /

注意：C 是大写的，C 后面有个空格，C 是改变解压安装目录的意思

说明： 执行以上命令，LogoMaker 将会被安装到/usr/sbin 目录下，它只有一个文件，安装完之后在命令行输入 logomaker 可出现如下界面，在后面的章节我们会介绍它的使用方法：


Friendly Linux Logo Maker
File Help
FriendlyARM
BE DIFFERENT

1.5 配置和编译 U-boot

三星已经为 6410 移植好了 U-boot，并且支持 USB 下载，Nand 启动等，它是开源的，仅此而已。我们在此基础上对 U-boot 做了诸多改进：

- 增加了下载菜单，类似 Superboot 的 USB 下载菜单
- 增加了 SD 卡启动配置
- 支持直接下载烧写 yaffs2 文件系统映像
- 支持烧写 WindowsCE BootLoader 之 Nboot
- 支持烧写 WindowsCE 映像的功能
- 支持烧写单文件映像文件，就是通常所说的裸机程序
- 支持返回原始 shell
- 增加了对 256M DDR RAM 的支持

下面我们就介绍一下它的配置和编译以及使用方法。

1.5.1 配置编译支持 NAND 启动的 U-boot

说明：根据开发板不同的内存(DDR RAM)容量，需要使用不同的 U-boot 配置项。

要编译适合于 128M 内存的 U-boot，请按照以下步骤：

进入 U-boot 源代码目录，执行：

#cd /opt/FriendlyARM/mini6410/linux/u-boot-mini6410

#make mini6410_nand_config-ram128;make

将会在当前目录配置并编译生成支持 Nand 启动的 U-boot.bin，使用 SD 卡或者 USB 下载到 Nand Flash 即可使用，详见“刷机指南”，光盘中 images/linux 目录中已经提供了编译好的该文件，为了便于区分，我们把它重新命名为 u-boot_nand-ram128.bin

要编译适合于 256M 内存的 U-boot，请按照以下步骤：

进入 U-boot 源代码目录，执行：

#cd /opt/FriendlyARM/mini6410/linux/u-boot-mini6410

#make mini6410_nand_config-ram256;make

将会在当前目录配置并编译生成支持 Nand 启动的 U-boot.bin，使用 SD 卡或者 USB 下载到 Nand Flash 即可使用，详见“刷机指南”，光盘中 images/linux 目录中已经提供了编译好的该文件，为了便于区分，我们把它重新命名为 u-boot_nand-ram256.bin

1.5.2 配置编译支持 SD 卡启动的 U-boot

说明：根据开发板不同的内存(DDR RAM)容量，需要使用不同的 U-boot 配置项。

要编译适合于 128M 内存的 U-boot，请按照以下步骤：

进入 U-boot 源代码目录，执行：

#cd /opt/FriendlyARM/mini6410/linux/u-boot-mini6410

#make mini6410_sd_config-ram128;make

将会在当前目录配置并编译生成支持 SD 启动的 U-boot.bin，使用 SD-Flasher.exe 工具把它烧写到 SD 卡中，设置开发板从 SD 卡启动即可使用了，可以参考 2.2 章节的步骤，只需把其中的 Superboot.bin 改为 U-boot.bin 就可以了。光盘中 images/linux 目录中已经提供了编译好的该文件，为了便于区分，我们把它重新命名为 u-boot_sd-ram128.bin

要编译适合于 256M 内存的 U-boot，请按照以下步骤：

进入 U-boot 源代码目录，执行：

#cd /opt/FriendlyARM/mini6410/linux/u-boot-mini6410

#make mini6410_sd_config-ram256;make

将会在当前目录配置并编译生成支持 SD 启动的 U-boot.bin，使用 SD-Flasher.exe 工具把它烧写到 SD 卡中，设置开发板从 SD 卡启动即可使用了，可以参考 2.2 章节的步骤，只需把其中的 Superboot.bin 改为 U-boot.bin 就可以了。光盘中 images/linux 目录中已经提供了编译好的该文件，为了便于区分，我们把它重新命名为 u-boot_sd-ram256.bin

1.5.3 U-boot 使用说明

未完

1.6 配置和编译内核(Kernel)

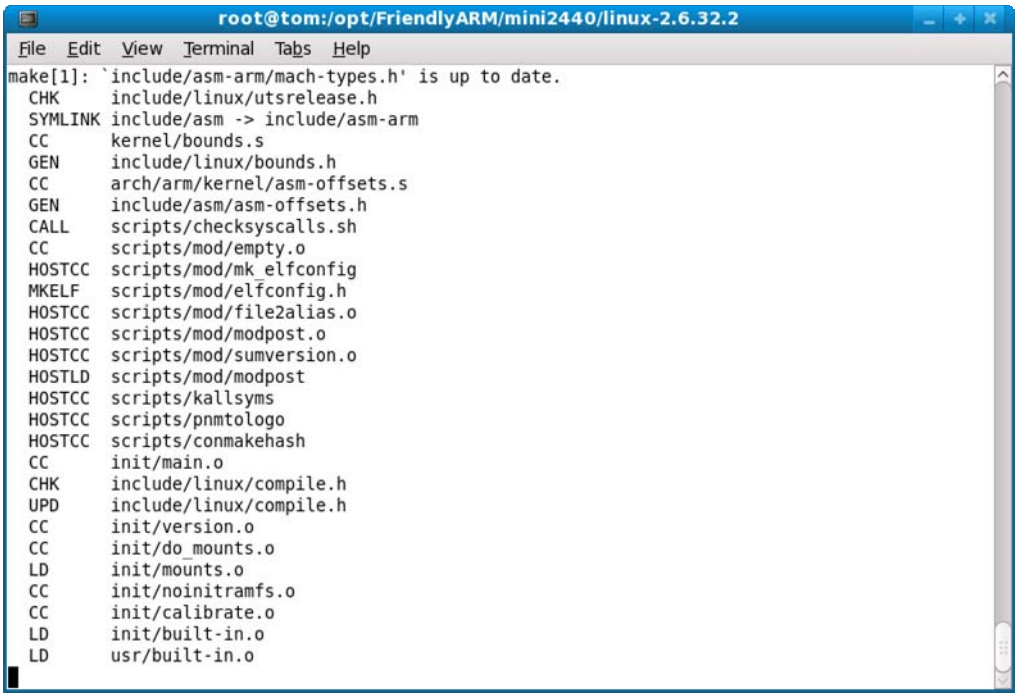
为了方便用户能够编译出和光盘烧写文件完全一致的内核，我们针对不同的 LCD 输出分别做了相应的内核配置文件：

config_mini6410_x35 –适用于 Sony 3.5”LCD，分辨率为 240x320
config_mini6410_n43 –适用于 NEC4.3”LCD，分辨率为 480x272
config_mini6410_l80 –适用于 Sharp 8”(或兼容)LCD，分辨率为 640x480
config_mini6410_a70 –适用于 7 寸真彩屏，分辨率为 800x480
config_mini6410_vga1024x768 –适用于分辨率输出为 1024x768 的 VGA 模块转接板
config_mini6410_vga800x600 –适用于分辨率输出为 800x600 的 VGA 模块转接板
config_mini6410_vga640x480 –适用于分辨率输出为 640x480 的 VGA 模块转接板
config_mini6410_ezvga800x600 –适用于简易 VGA 转接板，输出分辨率为 800x600

如下命令缺省了配置文件 config_n43 来编译内核

#cp config_mini6410_n43 .config　；注意：n43 后面有个空格，然后有个“.”开头的 config

#make zImage ;开始编译内核，也可以直接使用 make 命令



编译结束后，会在 **arch/arm/boot** 目录下生成 linux 内核映象文件 **zImage**，你可以使用第三章节介绍的方法把 zImage 下载到开发板测试。

光盘 images/linux 目录中已经提供好了编译好的内核文件，根据不同类型的 LCD 型号，分别有：zImage_n43, zImage_a70 等

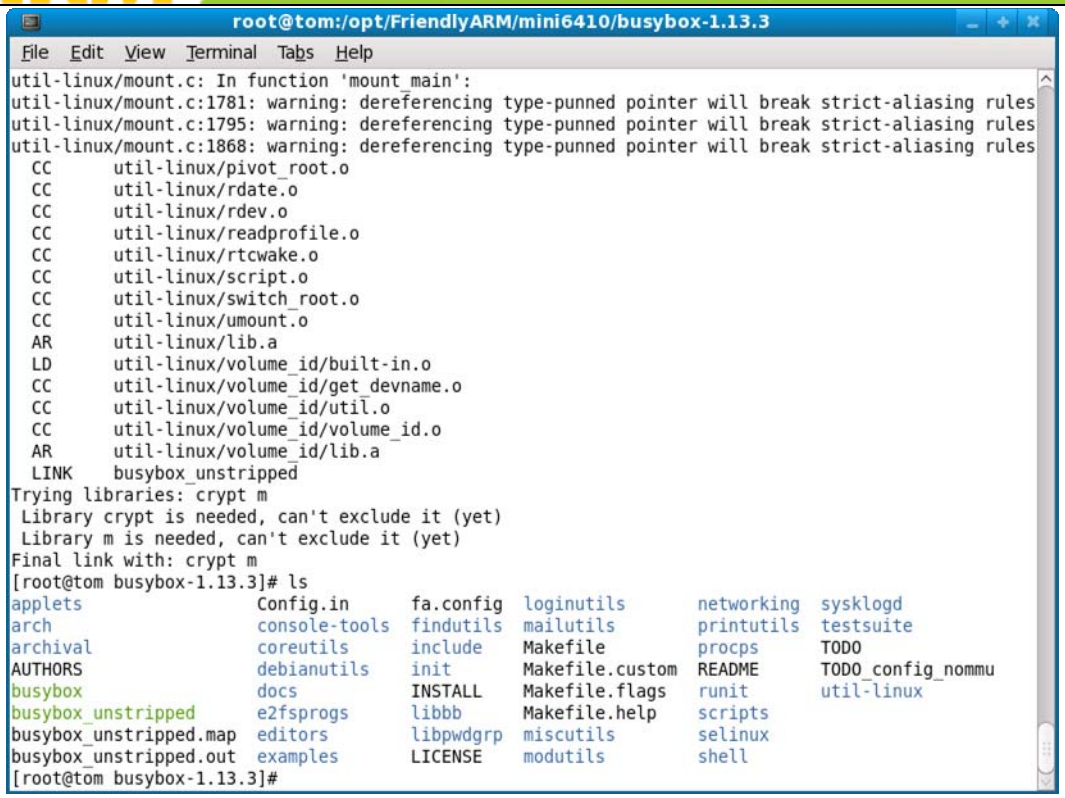
1.7 配置和编译 busybox

一般从官方网站下载的 busybox 源代码，需要根据所需重新配置一下，才可以编译使用，我们已经做好了一个缺省的配置文件：fa.config，无论是 2440 和 6410 我们均使用了此配置，通过它编译出的 busybox 可以满足绝大部分的需要，进入 busybox 源代码目录，执行：

#cp fa.config .config

#make

稍等一会，即可在当前目录编译生成 **busybox** 目标文件，它和开发板预装的是一样的，一般 busybox 是不需要更新的。



1.8 制作目标板文件系统映象

请确认你已经按照 4.4.4 一节的步骤安装了 mktools 系列工具，它们是将把同一个目标文件系统目录压制为不同格式的映像文件，以用来安装烧写到 Nand Flash 或者复制到 SD 卡中运行。

还请确认你已经做好了 4.4.2 的准备工作，也就是准备了目标文件系统目录。

1.8.1 制作 yaffs2 文件系统映像

进入工作目录/opt/FriendlyARM/mini6410/linux，执行以下命令：

#mkyaffs2image-128M　rootfs_qtopia_qt4　rootfs_qtopia_qt4.img

将把 rootfs_qtopia_qt4 目录压制为 yaffs2 格式的 rootfs_qtopia_qt4.img 映像文件，它和光盘中/images/Linux/目录下的同名文件是相同的，使用 SD 卡或者 USB 下载可以把它烧写到 Nand Flash 中，烧写步骤详见“刷机指南”。

说明：你也可以使用该命令工具把 rootfs_qtopia_qt4-s 目录压制为 yaffs2 映像文件，在此不再赘述。

1.8.2 制作 ubifs 文件系统映像

进入工作目录/opt/FriendlyARM/mini6410/linux，执行以下命令：

#mkyaffs2image-128M　rootfs_qtopia_qt4　rootfs_qtopia_qt4.ubi

将把 rootfs_qtopia_qt4 目录压制为 UBIFS 格式的 rootfs_qtopia_qt4.ubi 映像文件，它和光盘中/images/Linux/目录下的同名文件是相同的，使用 SD 卡或者 USB 下载可以把它烧写到 Nand Flash 中，烧写步骤详见“刷机指南”。

说明：你也可以使用该命令工具把 rootfs_qtopia_qt4-s 目录压制为 ubifs 映像文件，在此不再赘述

1.8.3 制作 ext3 文件系统映像

进入工作目录/opt/FriendlyARM/mini6410/linux，执行以下命令：

#mkext3image-128M　rootfs_qtopia_qt4　rootfs_qtopia_qt4.ext3

将把 rootfs_qtopia_qt4 目录压制为 EXT3 格式的 rootfs_qtopia_qt4.ext3 映像文件，它和光盘中/images/Linux/目录下的同名文件是相同的，参考 3.3 章节的说明，把它复制到 SD 卡就可以直接使用它运行系统了。

说明：你也可以使用该命令工具把 rootfs_qtopia_qt4-s 目录压制为 ext3 映像文件，在此不再赘述

1.9 嵌入式 Linux 应用程序示例

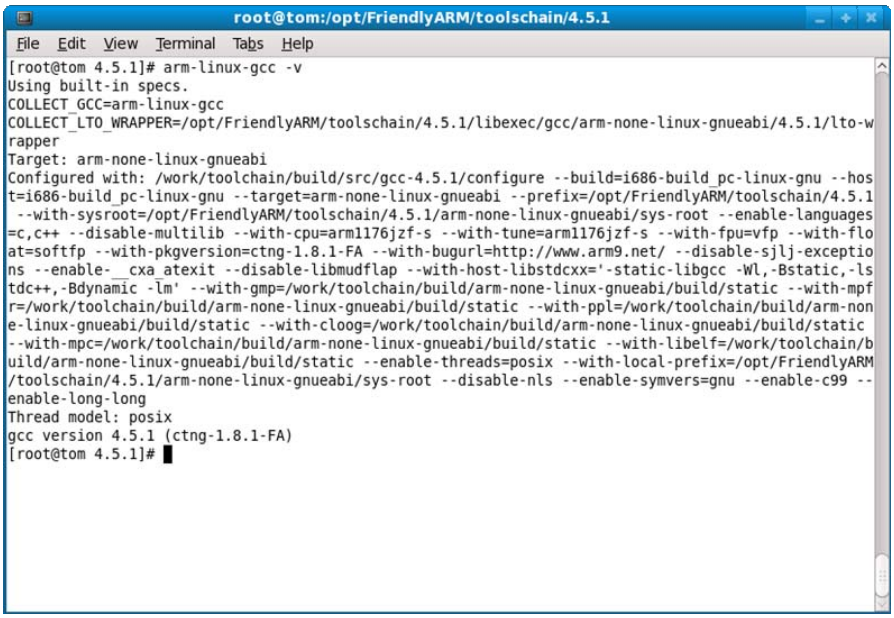
本节内容通过嵌入式 Linux 开发最简单的例子，介绍了如何编写和编译 Linux 应用程序，并下载到开发板运行起来。

如果你已经执行了 4.4.1 章节的步骤，就可以在**/opt/FriendlyARM/mini6410/examples** 目录下找到以下示例的源代码，实际可能更多一些，下面仅选几个典型的以作指引。

嵌入式 Linux 资源丰富，我们不可能介绍到每一个细节，本文旨在提供一些嵌入式 Linux 经常用到的方法，为你打开奇妙世界的大门。

注意：以下示例程序所使用的编译器为 arm-linux-gcc-4.5.1-v6-vfp，如果你使用了其他版本的交叉编译器，编译完有可能无法在开发板上运行。

要检查交叉编译器的版本类型，可在终端运行命令：arm-linux-gcc –v，如图



声明：以下 Linux 示例程序均为友善之臂原创，我们发现有的开发板厂商或者个人修改了 copyright 说明，据为己有，虽然国内对这种抄袭行为基本没有法律约束，但我们对这种无耻的盗窃行为予以鄙视，并敬告大家要尊重原厂家的辛苦劳动。

1.9.1 Hello,World!

Hello,World 源代码位于位于**/opt/FriendlyARM/mini6410/linux/examples/hello** 目录，其源代码如下：

```
#include <stdio.h>

int main(void) {
        printf("hello, FriendlyARM!\n");
}
```

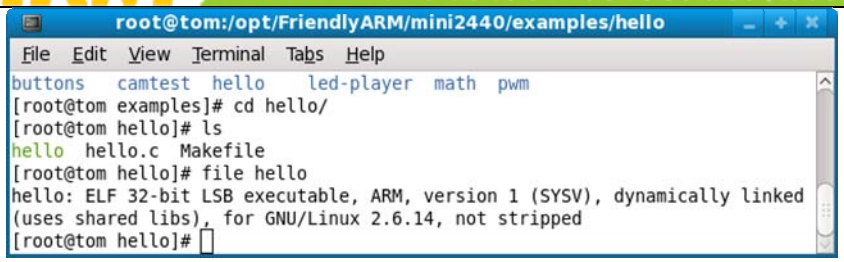
Step1:编译 Hello,World

进入源代码目录，并执行 make：

#cd /opt/FriendlyARM/mini6410/linux/examples/hello

#make

最后将生成 hello 可执行文件，使用 file 命令可以检查你生成的 hello 可执行文件是否为 ARM 体系和格式版本，能在开发板上正常运行的可执行文件一般如图所示：



Step2: 把 Hello,World 下载到开发板运行

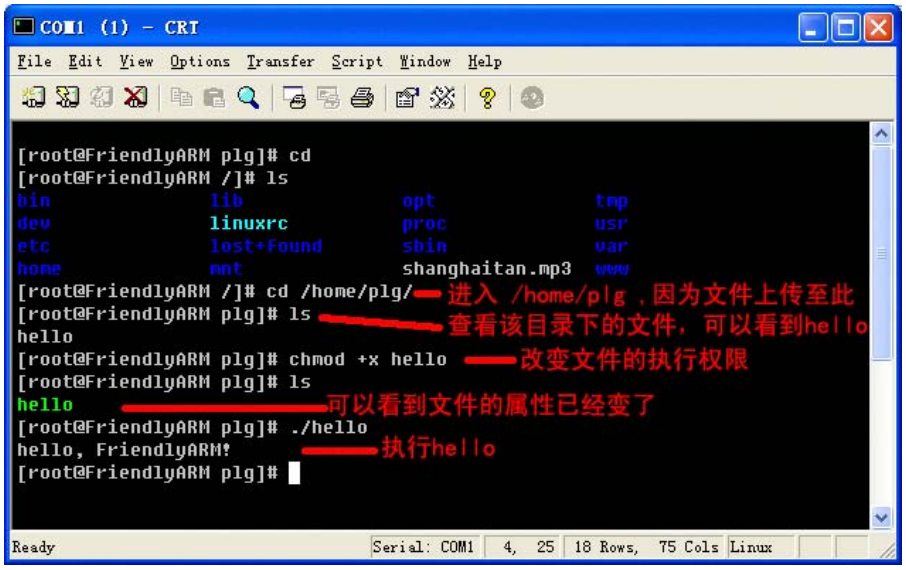
将编译好的可执行文件下载到目标板目前主要四种方式：
第一种：通过 ftp 传送文件到开发板(**推荐使用**)
第二种：复制到介质(如优盘)
第三种：通过串口传送文件到开发板
第四种：通过 NFS(网络文件系统)直接运行
下面分别进行介绍：

(1) 使用 ftp 传送文件(推荐使用)

说明：使用 ftp 登录目标板，把编译好的程序上传；然后修改上传后目标板上的程序的可执行属性，并执行。

首先，在 PC 端执行，如图所示

然后，在目标板一端执行，如图所示



(2) 使用优盘

说明：先把编译好的可执行程序复制到优盘，再把优盘插到目标板上并挂载它，然后把程序拷贝到目标板的可执行目录/bin

步骤：

1. 复制程序到优盘

把优盘插到 PC 的 USB 接口，执行以下命令把程序复制到优盘

#mount /dev/sda1 /mnt　　；挂接优盘

#cp hello /mnt　　；复制刚才编译好的程序到优盘

#umount /mnt　　；卸载优盘

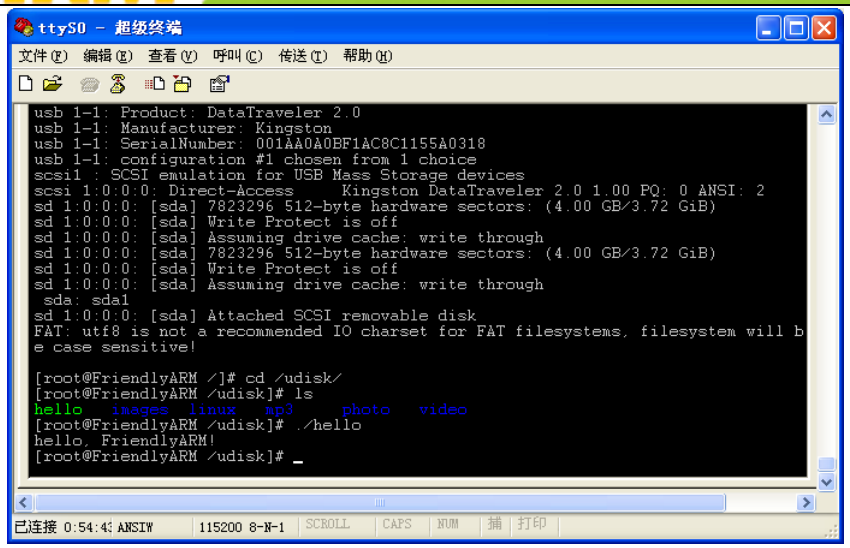
2. 把程序从优盘拷贝到目标板并执行

把优盘插入到开发板的 USB Host 接口，优盘会自动挂载到/udisk 目录，执行以下命令就可以运行 hello 程序了。

#cd /udisk

#./hello　　；执行 hello 程序

注意：如果此时强制拔出优盘，需要退回到根目录，再执行 umount /udisk 方可为下一次做好自动挂载的准备。



(3)通过串口传送文件到开发板

通过 4.2.5 章节我们学会了如何通过串口传送文件到开发板，你也可以通过相同的方法传送 hello 可执行程序，具体步骤在此不再详细描述，记得传送完毕把文件的属性改为可执行才能正常运行。

#chmod +x hello

说明：有些用户使用 USB 转串口线，因为有些转接器性能是不太好的，所以有时会出现“传输超时”或者根本无法传输到开发板的现象，因此我们建议使用 ftp 传送到开发板。

(4)通过网络文件系统 NFS 执行

Linux 中最常用的方法就是采用 NFS 来执行各种程序，这样可以不必花费很多时间下载程序，虽然在此下载 hello 程序用不了多久，一旦您的应用程序变得越来越大，您就会发现使用 NFS 运行的方便所在。

如同前面所讲述的那样，请先按照 4.3.4 一节搭建好 NFS 服务器系统，然后在命令行输入以下命令（假定服务器的 IP 地址为 192.168.1.111）：

#mount –t nfs –o nolock 192.168.1.111:/opt/FriendlyARM/mini6410/linux/rootfs_qtopia_qt4 /mnt

挂接成功，您就可以进入/mnt 目录进行操作了，在您的 PC Linux 终端把 hello 复制到 **opt/FriendlyARM/mini6410/linux/rootfs_qtopia_qt4** 目录，然后在开发板的串口终端执行

#cd /mnt

#./hello

1.9.2 LED 测试程序

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char

驱动程序名称	mini6410_leds.c
设备类型	misc
设备名	/dev/leds
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/leds
测试程序名称	led.c
测试程序可执行文件名称	led
测试程序在开发板中的位置	
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

程序清单

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
{
    int on;
    int led_no;
    int fd;

/*  检查 led 控制的两个参数，如果没有参数输入则退出。*/
    if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d", &on) != 1 ||
        on < 0 || on > 1 || led_no < 0 || led_no > 3) {
        fprintf(stderr, "Usage: leds led_no 0|1\n");
        exit(1);
    }

/*打开/dev/leds 设备文件*/
    fd = open("/dev/leds0", 0);
    if (fd < 0) {
        fd = open("/dev/leds", 0);
    }
    if (fd < 0) {
        perror("open device leds");
        exit(1);
    }

/*通过系统调用 ioctl 和输入的参数控制 led*/
    ioctl(fd, on, led_no);
/*关闭设备句柄*/
```

```
    close(fd);
    return 0;
}
```

你可以按照上面的 hello 程序的步骤编译出 led 可执行文件，然后下载到开发板运行它。

1.9.3 测试按键

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char
驱动程序名称	Mini6410_buttons.c
设备类型	misc
设备名	/dev/buttons
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/buttons
测试程序源代码名称	buttons_test.c
测试程序可执行文件名称	buttons
测试程序在开发板中的位置	
说明：按键驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

程序清单

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
{
    int buttons_fd;
    char buttons[6] = {'0', '0', '0', '0', '0', '0'};

    buttons_fd = open("/dev/buttons", 0);
    if (buttons_fd < 0) {
        perror("open device buttons");
```

```
            exit(1);
        }

        for (;;) {
            char current_buttons[6];
            int count_of_changed_key;
            int i;
            if (read(buttons_fd, current_buttons, sizeof current_buttons) != sizeof current_buttons) {
                perror("read buttons:");
                exit(1);
            }

            for (i = 0, count_of_changed_key = 0; i < sizeof buttons / sizeof buttons[0]; i++) {
                if (buttons[i] != current_buttons[i]) {
                    buttons[i] = current_buttons[i];
                    printf("%skey %d is %s", count_of_changed_key? ", ": "", i+1, buttons[i] ==
'0' ? "up" : "down");
                    count_of_changed_key++;
                }
            }
            if (count_of_changed_key) {
                printf("\n");
            }


        }

        close(buttons_fd);
        return 0;
}
```

你可以按照上面的 hello 程序的步骤手编译出 buttons 可执行文件，然后下载到开发板运行它

1.9.4 PWM 控制蜂鸣器编程示例

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char
驱动程序名称	Mini6410_pwm.c
设备类型	misc
设备名	/dev/pwm

测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/pwm
测试程序源代码名称	pwm_test.c
测试程序可执行文件名称	Pwm_test
测试程序在开发板中的位置	
说明：PWM 控制蜂鸣器驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载	

程序清单

```
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>

#define PWM_IOCTL_SET_FREQ          1
#define PWM_IOCTL_STOP              2

#define  ESC_KEY         0x1b

static int getch(void)
{
    struct termios oldt,newt;
    int ch;

    if (!isatty(STDIN_FILENO)) {
        fprintf(stderr, "this problem should be run at a terminal\n");
        exit(1);
    }
    // save terminal setting
    if(tcgetattr(STDIN_FILENO, &oldt) < 0) {
        perror("save the terminal setting");
        exit(1);
    }

    // set terminal as need
    newt = oldt;
    newt.c_lflag &= ~( ICANON | ECHO );
    if(tcsetattr(STDIN_FILENO,TCSANOW, &newt) < 0) {
        perror("set terminal");
        exit(1);
    }

    ch = getchar();
```

```
    // restore termial setting
    if(tcsetattr(STDIN_FILENO,TCSANOW,&oldt) < 0) {
        perror("restore the termial setting");
        exit(1);
    }
    return ch;
}

static int fd = -1;
static void close_buzzer(void);
static void open_buzzer(void)
{
    fd = open("/dev/pwm", 0);
    if (fd < 0) {
        perror("open pwm_buzzer device");
        exit(1);
    }

    // any function exit call will stop the buzzer
    atexit(close_buzzer);
}

static void close_buzzer(void)
{
    if (fd >= 0) {
        ioctl(fd, PWM_IOCTL_STOP);
        close(fd);
        fd = -1;
    }
}

static void set_buzzer_freq(int freq)
{
    // this IOCTL command is the key to set frequency
    int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, freq);
    if(ret < 0) {
        perror("set the frequency of the buzzer");
        exit(1);
    }
```

```
}
static void stop_buzzer(void)
{
	int ret = ioctl(fd, PWM_IOCTL_STOP);
	if(ret < 0) {
		perror("stop the buzzer");
		exit(1);
	}
}

int main(int argc, char **argv)
{
	int freq = 1000 ;

	open_buzzer();

	printf( "\nBUZZER TEST ( PWM Control )\n" );
	printf( "Press +/- to increase/reduce the frequency of the BUZZER\n" ) ;
	printf( "Press 'ESC' key to Exit this program\n\n" );


	while( 1 )
	{
		int key;

		set_buzzer_freq(freq);
		printf( "\tFreq = %d\n", freq );

		key = getch();

		switch(key) {
		case '+':
			if( freq < 20000 )
				freq += 10;
			break;

		case '-':
			if( freq > 11 )
				freq -= 10 ;
			break;
```

```
            case ESC_KEY:
            case EOF:
                stop_buzzer();
                exit(0);

            default:
                break;
            }
        }
}
```

你可以按照上面的 hello 程序的步骤手编译出 buttons 可执行文件，然后下载到开发板运行它

1.9.5 I2C-EEPROM 编程示例

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/i2c/busses
驱动程序名称	I2c-s3c2410.c
设备类型	字符设备
设备名	/dev/i2c/0
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/i2c
测试程序源代码名称	eeprog.c 24cXX.c
测试程序可执行文件名称	i2c
测试程序在开发板中的位置	
说明：I2C 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	
程序清单	
注意：以下程序还需同目录下 24cXX.c 程序的支持	

```
#include <stdio.h>
#include <fcntl.h>
#include <getopt.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "24cXX.h"
```

```
#define usage_if(a) do { do_usage_if( a , __LINE__); } while(0);
void do_usage_if(int b, int line)
{
	const static char *eeprog_usage =
		"I2C-24C08(256 bytes) Read/Write Program, ONLY FOR TEST!\n"
		"FriendlyARM Computer Tech. 2009\n";
	if(!b)
		return;
	fprintf(stderr, "%s\n[line %d]\n", eeprog_usage, line);
	exit(1);
}


#define die_if(a, msg) do { do_die_if( a , msg, __LINE__); } while(0);
void do_die_if(int b, char* msg, int line)
{
	if(!b)
		return;
	fprintf(stderr, "Error at line %d: %s\n", line, msg);
	fprintf(stderr, "	sysmsg: %s\n", strerror(errno));
	exit(1);
}


static int read_from_eeprom(struct eeprom *e, int addr, int size)
{
	int ch, i;
	for(i = 0; i < size; ++i, ++addr)
	{
		die_if((ch = eeprom_read_byte(e, addr)) < 0, "read error");
		if( (i % 16) == 0 )
			printf("\n %.4x|    ", addr);
		else if( (i % 8) == 0 )
			printf("    ");
		printf("%.2x ", ch);
		fflush(stdout);
	}
	fprintf(stderr, "\n\n");
	return 0;
}
```

```
static int write_to_eeprom(struct eeprom *e, int addr)
{
	int i;
	for(i=0, addr=0; i<256; i++, addr++)
	{
		if( (i % 16) == 0 )
			printf("\n %.4x|    ", addr);
		else if( (i % 8) == 0 )
			printf("    ");
		printf("%.2x ", i);
		fflush(stdout);
		die_if(eeprom_write_byte(e, addr, i), "write error");
	}
	fprintf(stderr, "\n\n");
	return 0;
}

int main(int argc, char** argv)
{
	struct eeprom e;
	int op;

	op = 0;

	usage_if(argc != 2 || argv[1][0] != '-' || argv[1][2] != '\0');
	op = argv[1][1];

	fprintf(stderr, "Open /dev/i2c/0 with 8bit mode\n");
	die_if(eeprom_open("/dev/i2c/0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e) < 0,
			"unable to open eeprom device file "
			"(check that the file exists and that it's readable)");
	switch(op)
	{
	case 'r':
		fprintf(stderr, "    Reading 256 bytes from 0x0\n");
		read_from_eeprom(&e, 0, 256);
		break;
	case 'w':
		fprintf(stderr, "    Writing 0x00-0xff into 24C08 \n");
```

```
            write_to_eeprom(&e, 0);
            break;
        default:
            usage_if(1);
            exit(1);
        }
        eeprom_close(&e);

        return 0;
}
```

1.9.6 串口编程示例

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/serial/
驱动程序名称	S3c6400.c
设备名	/dev/ttySAC0,1,2,4
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/comtest
测试程序源代码名称	comtest.c
测试程序可执行文件名称	armcomtest
测试程序在开发板中的位置	
说明：测试程序编译后可得到 x86 版本和 arm 版本，其源代码是完全一样的	

程序清单

说明：comtest 程序是友善之臂早期开发的一个串口测试程序，它其实是一个十分简易的串口终端程序，类似于 linux 中的 minicom，该程序与硬件无关，因此相同的代码不仅适用于任何 Arm-linux 开发板平台，也可以在 PC linux 上运行使用，方法都是完全一样的。通过该程序你可以了解串口编程的一些常见关键设置，对于 linux 下串口编程很有帮助和借鉴意义，该程序虽然十分短小，但设计极为严谨巧妙，我们对此就不详细解释了，下面是它的完整源代码：


```
# include <stdio.h>
# include <stdlib.h>
# include <termio.h>
# include <unistd.h>
# include <fcntl.h>
# include <getopt.h>
# include <time.h>
```

```
# include <errno.h>
# include <string.h>

static void Error(const char *Msg)
{
        fprintf (stderr, "%s\n", Msg);
        fprintf (stderr, "strerror() is %s\n", strerror(errno));
        exit(1);
}
static void Warning(const char *Msg)
{
         fprintf (stderr, "Warning: %s\n", Msg);
}


static int SerialSpeed(const char *SpeedString)
{
        int SpeedNumber = atoi(SpeedString);
#       define TestSpeed(Speed) if (SpeedNumber == Speed) return B##Speed
        TestSpeed(1200);
        TestSpeed(2400);
        TestSpeed(4800);
        TestSpeed(9600);
        TestSpeed(19200);
        TestSpeed(38400);
        TestSpeed(57600);
        TestSpeed(115200);
        TestSpeed(230400);
        Error("Bad speed");
        return -1;
}

static void PrintUsage(void)
{

     fprintf(stderr, "comtest - interactive program of comm port\n");
     fprintf(stderr, "press [ESC] 3 times to quit\n\n");

     fprintf(stderr, "Usage: comtest [-d device] [-t tty] [-s speed] [-7] [-c] [-x] [-o] [-h]\n");
     fprintf(stderr, "                -7 7 bit\n");
```

```
    fprintf(stderr, "               -x hex mode\n");
    fprintf(stderr, "               -o output to stdout too\n");
    fprintf(stderr, "               -c stdout output use color\n");
    fprintf(stderr, "               -h print this help\n");
    exit(-1);
}

static inline void WaitFdWriteable(int Fd)
{
      fd_set WriteSetFD;
      FD_ZERO(&WriteSetFD);
      FD_SET(Fd, &WriteSetFD);
      if (select(Fd + 1, NULL, &WriteSetFD, NULL, NULL) < 0) {
        Error(strerror(errno));
      }

}

int main(int argc, char **argv)
{
      int CommFd, TtyFd;

      struct termios TtyAttr;
      struct termios BackupTtyAttr;

      int DeviceSpeed = B115200;
      int TtySpeed = B115200;
      int ByteBits = CS8;
      const char *DeviceName = "/dev/ttyS0";
      const char *TtyName = "/dev/tty";
      int OutputHex = 0;
      int OutputToStdout = 0;
      int UseColor = 0;

      opterr = 0;
      for (;;) {
            int c = getopt(argc, argv, "d:s:t:7xoch");
            if (c == -1)
                  break;
            switch(c) {
```

```
            case 'd':
                    DeviceName = optarg;
                    break;
            case 't':
                    TtyName = optarg;
                    break;
            case 's':
           if (optarg[0] == 'd') {
          DeviceSpeed = SerialSpeed(optarg + 1);
           } else if (optarg[0] == 't') {
          TtySpeed = SerialSpeed(optarg + 1);
           } else
                      TtySpeed = DeviceSpeed = SerialSpeed(optarg);
                    break;
    case 'o':
           OutputToStdout = 1;
           break;
    case '7':
           ByteBits = CS7;
           break;
            case 'x':
                    OutputHex = 1;
                    break;
    case 'c':
           UseColor = 1;
           break;
            case '?':
            case 'h':
            default:
           PrintUsage();
            }
     }
     if (optind != argc)
            PrintUsage();

     CommFd = open(DeviceName, O_RDWR, 0);
     if (CommFd < 0)
    Error("Unable to open device");
     if (fcntl(CommFd, F_SETFL, O_NONBLOCK) < 0)
          Error("Unable set to NONBLOCK mode");
```

```
    memset(&TtyAttr, 0, sizeof(struct termios));
    TtyAttr.c_iflag = IGNPAR;
    TtyAttr.c_cflag = DeviceSpeed | HUPCL | ByteBits | CREAD | CLOCAL;
    TtyAttr.c_cc[VMIN] = 1;

    if (tcsetattr(CommFd, TCSANOW, &TtyAttr) < 0)
        Warning("Unable to set comm port");

    TtyFd = open(TtyName, O_RDWR | O_NDELAY, 0);
    if (TtyFd < 0)
   Error("Unable to open tty");

    TtyAttr.c_cflag = TtySpeed | HUPCL | ByteBits | CREAD | CLOCAL;
    if (tcgetattr(TtyFd, &BackupTtyAttr) < 0)
   Error("Unable to get tty");

    if (tcsetattr(TtyFd, TCSANOW, &TtyAttr) < 0)
   Error("Unable to set tty");


    for (;;) {
   unsigned char Char = 0;
   fd_set ReadSetFD;

   void OutputStdChar(FILE *File) {
        char Buffer[10];
        int Len = sprintf(Buffer, OutputHex ? "%.2X    " : "%c", Char);
        fwrite(Buffer, 1, Len, File);
   }

FD_ZERO(&ReadSetFD);

   FD_SET(CommFd, &ReadSetFD);
   FD_SET( TtyFd, &ReadSetFD);
#  define max(x,y) ( ((x) >= (y)) ? (x) : (y) )
   if (select(max(CommFd, TtyFd) + 1, &ReadSetFD, NULL, NULL, NULL) < 0) {
        Error(strerror(errno));
```

```
    }
#   undef max

    if (FD_ISSET(CommFd, &ReadSetFD)) {
        while (read(CommFd, &Char, 1) == 1) {

        WaitFdWriteable(TtyFd);
        if (write(TtyFd, &Char, 1) < 0) {
            Error(strerror(errno));
        }
        if (OutputToStdout) {
            if (UseColor)
            fwrite("\x1b[01;34m", 1, 8, stdout);
            OutputStdChar(stdout);
            if (UseColor)
            fwrite("\x1b[00m", 1, 8, stdout);
            fflush(stdout);
        }
        }
    }

    if (FD_ISSET(TtyFd, &ReadSetFD)) {
        while (read(TtyFd, &Char, 1) == 1) {
                static int EscKeyCount = 0;
        WaitFdWriteable(CommFd);
                if (write(CommFd, &Char, 1) < 0) {
            Error(strerror(errno));
        }
        if (OutputToStdout) {
            if (UseColor)
            fwrite("\x1b[01;31m", 1, 8, stderr);
            OutputStdChar(stderr);
            if (UseColor)
            fwrite("\x1b[00m", 1, 8, stderr);
            fflush(stderr);
            }

            if (Char == '\x1b') {
                        EscKeyCount ++;
                        if (EscKeyCount >= 3)
```

```
                                goto ExitLabel;
                        } else
                                EscKeyCount = 0;
                }
                 }

        }

ExitLabel:
        if (tcsetattr(TtyFd, TCSANOW, &BackupTtyAttr) < 0)
        Error("Unable to set tty");

        return 0;
}
```

1.9.7 UDP 网络编程

程序源代码说明	
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/net/
驱动程序名称	dm9000.c
该驱动的主设备号	无
设备名	eth0 (网络设备并不在/dev 目录中出现)
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/udptak
测试程序源代码名称	udptalk.c
测试程序可执行文件名称	udptalk.c
说明：测试程序编译后可得到 x86 版本和 arm 版本，其源代码是完全一样的	

程序原理分析

TCP/IP 提供了无连接的传输层协议：UDP(User Datagram Protocol，即用户数据报协议)。UDP 与 TCP 有很大的区别，因为无连接的 socket 编程与面向连接的 socket 编程也有很大的差异。由于不用建立连接，因此每个发送个接收的数据报都包含了发送方和接收方的地址信息。

在发送和接收数据之前，先要建立一个数据报方式的套接字，该 socket 的类型为 SOCK_DGRAM，用如下的调用产生：

sockfd=socket(AF_INET, SOCK_DGRAM, 0);

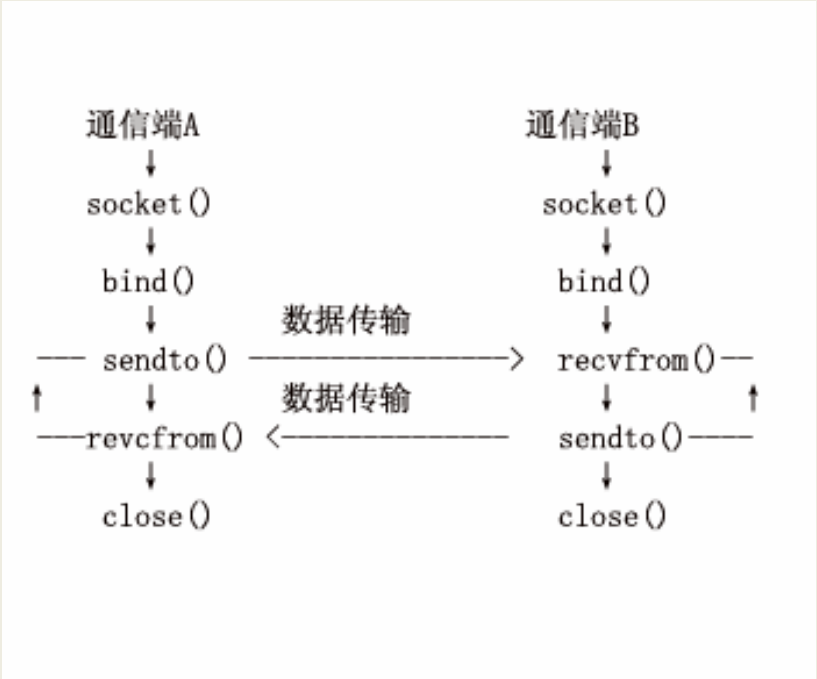
由于不需要建立连接，因此产生 socket 后就可以直接发送和接收了。当然，要接收数据报也必须绑定一个端口，否则发送方无法得知要发送到哪个端口。Sendto 和 recvfrom 两个系统调用分别用于发送和接收数据报，其调用格式为：

int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

int recvfrom(int, s, void *buf, int len, unsigned int flags, struct sockaddr *from, int fromlen);

其中 s 为所使用的 socket，msg 和 buf 分别为发送和接收的缓冲区指针，len 为缓冲区的长度，flags 为选项标志，此处还用不到，设为 0 即可。to 和 from 就是发送的目的地址和接收的来源地址，包含了 IP 地址和端口信息。tolen 和 fromlen 分别是 to 和 from 这两个 socket 地址结构的长度。这两个函数的返回值就是实际发送和接收的字节数，返回-1 表示出错。

使用无连接方式通信的基本过程如图所示。



UDP 通信的基本过程

上图描述的是通信双方都绑定自己地址端口的情形，但在某些情况下，也可能有一方不用绑定地址和端口。不绑定的一方的地址和端口由内核分配。由于对方无法预先知道不绑定的一方的端口和 IP 地址(假设主机有多个端口，这些端口分配了不同的 IP 地址)，因此只能由不绑定的一方先发出数据报，对方根据收到的数据报中的来源地址就可以确定回送数据报所需要的发送地址了。显然，在这种情况下对方必须绑定地址和端口，并且通信只能由非绑定方发起。

与 read()和 write()相似，进程阻塞在 recvfrom()和 sendto()中也会发生。但与 TCP 方式不同的是，接收到一个字节数为 0 的数据报是有可能的，应用程序完全可以将 sendto()中的 msg 设为 NULL，同时将 len 设为 0。

程序清单

```
/*
 *      udptalk ： Example for Matrix V    ;说明：本程序同样适用于 mini2440
 *
 *      Copyright (C) 2004 capbily - friendly-arm
 *       capbily@hotmail.com
 */
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <stdio.h>

#define BUFLEN 255

int main(int argc, char **argv)
{
    struct sockaddr_in peeraddr, /*存放谈话对方 IP 和端口的 socket 地址*/
                    localaddr;/*本端 socket 地址*/
    int sockfd;
    char recmsg[BUFLEN+1];
    int socklen, n;

    if(argc!=5){
        printf("%s <dest IP address> <dest port> <source IP address> <source port>\n",
argv[0]);
        exit(0);
    }

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd<0){
        printf("socket creating err in udptalk\n");
        exit(1);
    }
    socklen = sizeof(struct sockaddr_in);
    memset(&peeraddr, 0, socklen);
    peeraddr.sin_family=AF_INET;
    peeraddr.sin_port=htons(atoi(argv[2]));
    if(inet_pton(AF_INET, argv[1], &peeraddr.sin_addr)<=0){
        printf("Wrong dest IP address!\n");
        exit(0);
    }
    memset(&localaddr, 0, socklen);
    localaddr.sin_family=AF_INET;
    if(inet_pton(AF_INET, argv[3], &localaddr.sin_addr)<=0){
        printf("Wrong source IP address!\n");
        exit(0);
    }
    localaddr.sin_port=htons(atoi(argv[4]));
    if(bind(sockfd, &localaddr, socklen)<0){
```

```
        printf("bind local address err in udptalk!\n");
        exit(2);
    }

    if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
    if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
        printf("sendto err in udptalk!\n");
        exit(3);
    }

    for(;;){
        /*recv&send message loop*/
        n = recvfrom(sockfd, recmsg, BUFLEN, 0, &peeraddr, &socklen);
        if(n<0){
            printf("recvfrom err in udptalk!\n");
            exit(4);
        }else{
            /*成功接收到数据报*/
            recmsg[n]=0;
            printf("peer:%s", recmsg);
        }
        if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
        if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
            printf("sendto err in udptalk!\n");
            exit(3);
        }
    }
}
```

测试

将 udptalk.c 编译好后就可以运行了，/opt/FriendlyARM/mini6410/linux/examples/udptalk 目录下的 Makefile 指定了两个编译目标可执行文件，一个用于在主机端的 x86-udptalk，一个是用于开发板的 arm-udptalk，运行 make 命令将把这两个程序一起编译出来。可以把 arm-udptalk 使用上面介绍的方法下载到开发板中(预装的 Linux 不含该程序)，假设主机的 IP 地址为 192.168.1.108，开发板的 IP 地址为 192.168.1.230。

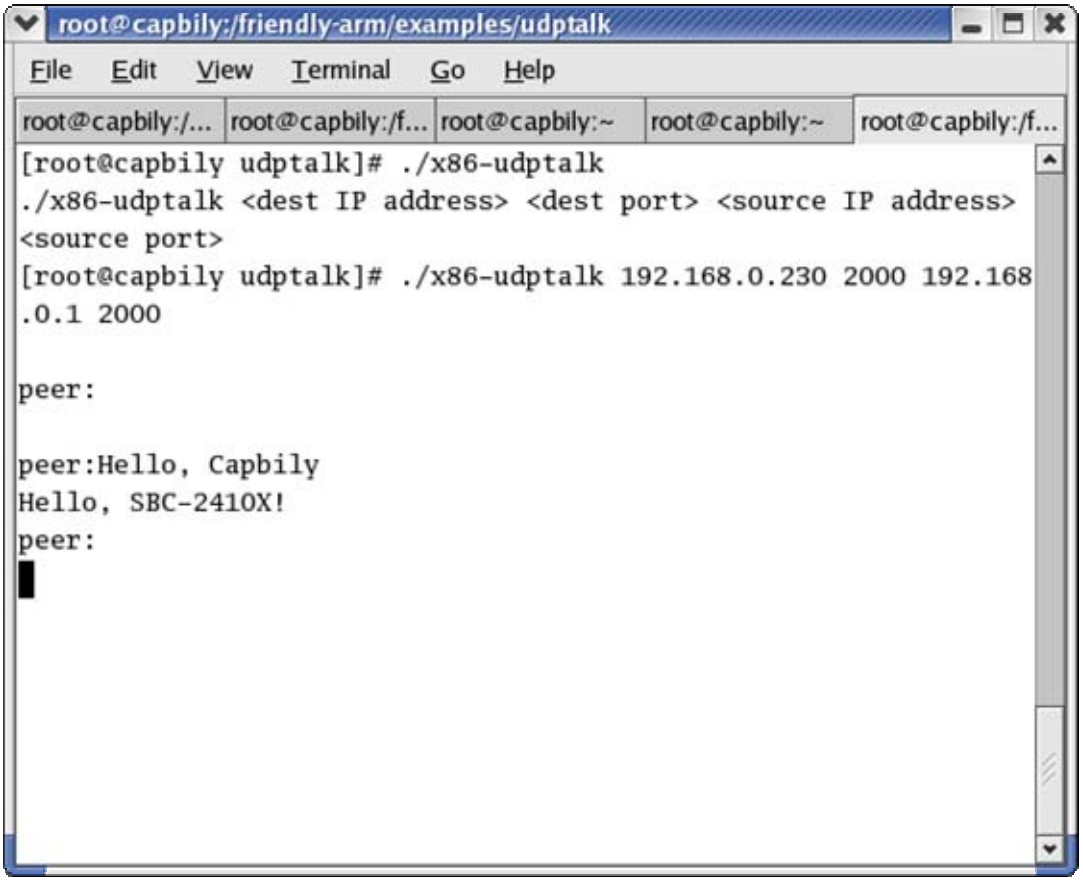
在主机的终端上输入：

#./x86-udptalk 192.168.1.230 2000 192.168.1.108 2000

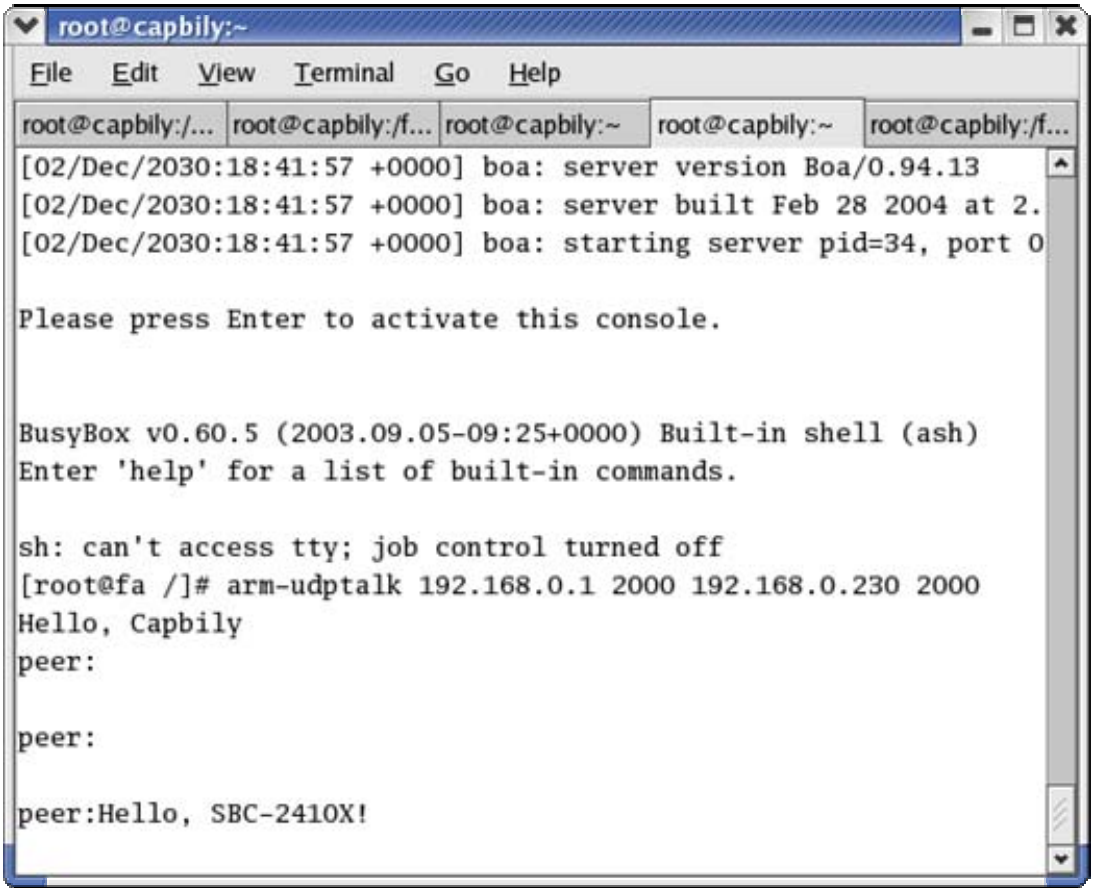
在开发板上的终端输入

#arm-udptalk 192.168.1.108 2000 192.168.1.230 2000

则运行结果分别如图所示：



在主机上运行 **x86-udptalk**



在开发板上运行 **arm-udptalk**

1.9.8 数学函数库调用示例

程序源代码说明	
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/math
测试程序源代码名称	mathtest.c
测试程序可执行文件名称	mathtest
程序清单	
注意：使用数学函数的关键是要包含其头文件 math.h，并且在编译的时候加入数学函数库 libm。	

#include <stdio.h>
#include <stdlib.h>
#include <math.h> **;注意：一定要包含此头文件**
 	
int main(void)
{
　　double a=8.733243;	

```
        printf("sqrt(%f)=%f\n", a, sqrt(a));

        return 0;
}
```

Makefile 内容：

```
CROSS=arm-linux-

all: mathtest

#注意：该处包含了数学函数库 libm，红色部分
mathtest:
        $(CROSS)gcc -o mathtest main.c -lm

clean:
        @rm -vf mathtest *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出 mathtest 可执行文件，然后下载到开发板运行它

1.9.9 线程编程示例

程序源代码说明	
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/pthread
测试程序源代码名称	pthread_test.c
测试程序可执行文件名称	pthread_test

程序清单

注意：使用线程的关键是要包含其头文件 pthread.h，并且在编译的时候加入线程库 libpthread。

```
#include<stddef.h>
#include<stdio.h>
#include<unistd.h>
#include"pthread.h" ;注意：一定要包含此头文件

void reader_function(void);
void writer_function(void);
char buffer;
int buffer_has_item=0;
pthread_mutex_t mutex;
main()
```

```
{
	pthread_t reader;
	pthread_mutex_init(&mutex,NULL);
	pthread_create(&reader,NULL,(void*)&reader_function,NULL);
	writer_function();
}
void writer_function(void)
{
	while(1)
	{
		pthread_mutex_lock(&mutex);
		if(buffer_has_item==0)
		{
			buffer='a';
			printf("make a new item\n");
			buffer_has_item=1;
		}
		pthread_mutex_unlock(&mutex);
	}
}
void reader_function(void)
{
	while(1)
	{
		pthread_mutex_lock(&mutex);
		if(buffer_has_item==1)
		{
			buffer='\0';
			printf("consume    item\n");
			buffer_has_item=0;
		}
		pthread_mutex_unlock(&mutex);
	}
}
```

Makefile 内容

```
CROSS=arm-linux-

all: pthread

#注意：该处包含了线程库 libphread，红色部分
```

```
pthread:
	$(CROSS)gcc -static -o pthread main.c -lpthread

clean:
	@rm -vf pthread *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出 pthread 可执行文件，然后下载到开发板运行它

1.9.10 管道应用编程示例-网页控制 LED

程序源代码说明	
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/led-player
测试程序源代码名称	led-player.c
测试程序可执行文件名称	led-player

原理说明

开机后我们可以通过网页发送命令控制开发板上的 LED 闪烁模式，其实这是进程间通信共享资源的一个典型例子，进程间通信就是 IPC(InterProcess Communication)，进程间通信的目的一般有：

(1)数据传输
(2)共享数据
(3)通知事件
(4)资源共享
(5)进程控制.

Linux 支持多种 IPC 机制，信号和管道是其中的两个。关于更详细的进程间通信的介绍，一般 Linux 编程的书上都有介绍，在此我们不多说了。

通过网页来控制 LED 的闪烁模式就是通过管道机制来实现的，其中 LED 是共享资源，led-player 是一个后台程序，它启动的时候就创建了一个命名管道/tmp/ led-control(当然该管道也可以通过命令 mknod 来创建，那样程序就要改写了，有兴趣的可以自己试试)， 并一直监测输入该管道的数据，根据不同的参数(模式 type 和周期 period)来改变 LED 的显示模式；leds.cgi 是一个网关程序，它接收从网页发送过来的字符形式指令（ping 代表跑马灯模式或者乒乓模式，counter 代表计数器模式，stop 代表停止模式，slow 代表周期为 0.25m，normal 代表周期为 0.125m，fast 代表周期为 0.0625m），并对这些指令进行赋值转换为实际数字，然后调用 echo 命令输送到管道/tmp/ led-control 以此实现对 LED 的控制，以下是各自的程序清单。

程序清单

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <sys/select.h>
#include <sys/time.h>
#include <string.h>

static int led_fd;
static int type = 1;

static void push_leds(void)
{
	static unsigned step;
	unsigned led_bitmap;
	int i;

	switch(type) {
	case 0:
		if (step >= 6) {
			step = 0;
		}
		if (step < 3) {
			led_bitmap = 1 << step;
		} else {
			led_bitmap = 1 << (6 - step);
		}
		break;
	case 1:
		if (step > 255) {
			step = 0;
		}
		led_bitmap = step;
		break;
	default:
		led_bitmap = 0;
	}
	step++;
	for (i = 0; i < 4; i++) {
		ioctl(led_fd, led_bitmap & 1, i);
		led_bitmap >>= 1;
	}
}
```

```
int main(void)
{
    int led_control_pipe;
    int null_writer_fd; // for read endpoint not blocking when control process exit

    double period = 0.5;

    led_fd = open("/dev/leds0", 0);
    if (led_fd < 0) {
        led_fd = open("/dev/leds", 0);
    }
    if (led_fd < 0) {
        perror("open device leds");
        exit(1);
    }
    unlink("/tmp/led-control");
    mkfifo("/tmp/led-control", 0666);

    led_control_pipe = open("/tmp/led-control", O_RDONLY | O_NONBLOCK);
    if (led_control_pipe < 0) {
        perror("open control pipe for read");
        exit(1);
    }
    null_writer_fd = open("/tmp/led-control", O_WRONLY | O_NONBLOCK);
    if (null_writer_fd < 0) {
        perror("open control pipe for write");
        exit(1);
    }

    for (;;) {
        fd_set rds;
        struct timeval step;
        int ret;

        FD_ZERO(&rds);
        FD_SET(led_control_pipe, &rds);
        step.tv_sec    = period;
        step.tv_usec = (period - step.tv_sec) * 1000000L;

        ret = select(led_control_pipe + 1, &rds, NULL, NULL, &step);
```

```
        if (ret < 0) {
            perror("select");
            exit(1);
        }
        if (ret == 0) {
            push_leds();
        } else if (FD_ISSET(led_control_pipe, &rds)) {
            static char buffer[200];
            for (;;) {
                char c;
                int len = strlen(buffer);
                if (len >= sizeof buffer - 1) {
                    memset(buffer, 0, sizeof buffer);
                    break;
                }
                if (read(led_control_pipe, &c, 1) != 1) {
                    break;
                }
                if (c == '\r') {
                    continue;
                }
                if (c == '\n') {
                    int tmp_type;
                    double tmp_period;
                    if (sscanf(buffer,"%d%lf", &tmp_type, &tmp_period) == 2) {
                        type = tmp_type;
                        period = tmp_period;
                    }
                    fprintf(stderr, "type is %d, period is %lf\n", type, period);
                    memset(buffer, 0, sizeof buffer);
                    break;
                }
                buffer[len] = c;
            }
        }
    }

    close(led_fd);
    return 0;
}
```

使用 make 指令可以直接编译出 led-player 可执行文件，它被作为一个服务器放置在开发板的/sbin 目录中。

Leds.cgi 网关程序源代码(该程序在开发板上的位置：/www/leds.cgi)，可见该网关程序其实就是一个 shell 脚本，它被网页 leds.html 调用为一个执行“action”，该脚本清单如下：

Leds.cgi 脚本清单

```
#!/bin/sh

type=0
period=1

case $QUERY_STRING in
        *ping*)
                type=0
                ;;
        *counter*)
                type=1
                ;;
        *stop*)
                type=2
                ;;
esac

case $QUERY_STRING in
        *slow*)
                period=0.25
                ;;
        *normal*)
                period=0.125
                ;;
        *fast*)
                period=0.0625
                ;;
esac

/bin/echo $type $period > /tmp/led-control

echo "Content-type: text/html; charset=gb2312"
```

```
echo
/bin/cat led-result.template

exit 0
```

1.9.11 基于 C++的 Hello,World

程序源代码说明	
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/c++
测试程序源代码名称	cplus.c++
测试程序可执行文件名称	cplus

程序清单

```
#include <iostream>
#include <cstring>
using namespace std;

class String
{
private:
            char *str;
public:
       String(char *s)
       {
              int lenght=strlen(s);
              str = new char[lenght+1];
              strcpy(str, s);
       }
       ~String()
       {
      cout << "Deleting str.\n";
      delete[] str;
       }
       void display()
       {
      cout << str <<endl;
       }
};

int main(void)
{
```

```
String s1="I like FriendlyARM.";
cout << "s1=";
s1.display();
return 0;
double num, ans;

cout << "Enter num:";
}
```

你可以按照上面的 hello 程序的步骤手工编译出 cplus 可执行文件，然后下载到开发板运行它

1.10 嵌入式 Linux 驱动程序示例

上一节我们介绍了一个简单的 Linux 程序 Hello,World，它是运行于用户态的应用程序，现在我们先从一个运行于内核态的 Hello, World 程序开始，介绍驱动程序的编写和使用。

1.10.1 Hello,Module-最简单的嵌入式 Linux 驱动程序模块

程序源代码说明	
源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char
源代码文件名称	Mini6410_hello_module.c
该驱动的主设备号	无
设备名	无
测试程序源代码目录	无
测试程序名称	无
测试程序可执行文件名称	无
说明：该驱动装载后不会在 dev 下创建任何设备节点。	

程序清单

```
#include <linux/kernel.h>
#include <linux/module.h>


static int __init mini6410_hello_module_init(void)
{
    printk("Hello, Mini6410 module is installed !\n");
    return 0;
}
```

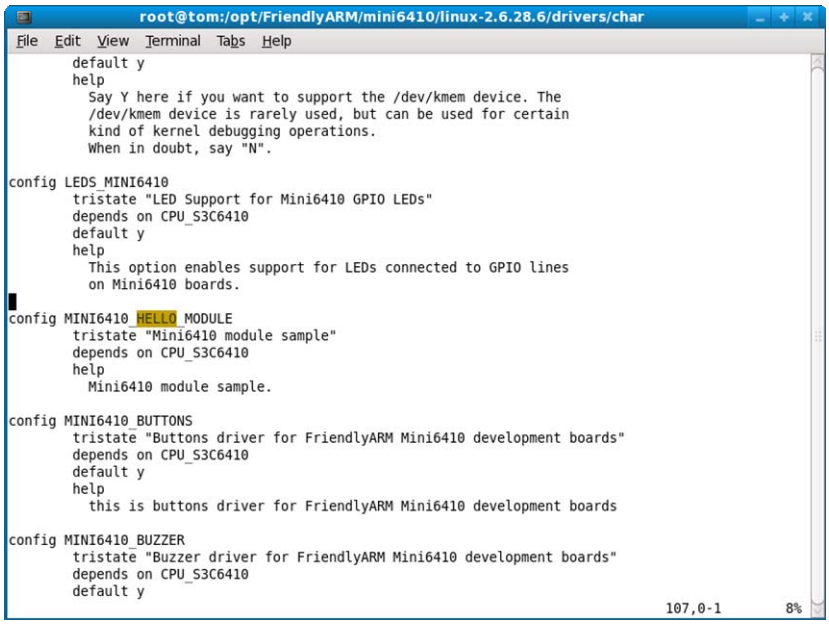
```
static void __exit mini6410_hello_module_cleanup(void)
{
	printk("Good-bye, Mini6410 module was removed!\n");
}

module_init(mini6410_hello_module_init);
module_exit(mini6410_hello_module_cleanup);
MODULE_LICENSE("GPL");
```

(1)把 Hello,Module 加入内核代码树，并编译

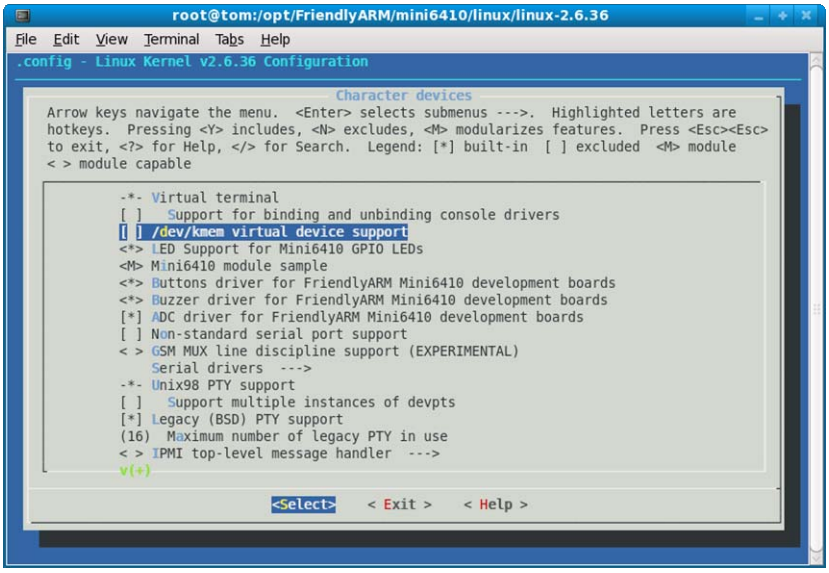
一般编译 2.6 版本的驱动模块需要把驱动代码加入内核代码树，并做相应的配置，如下步骤(注意：实际上以下步骤均已经做好，你只需要打开检查一下直接编译就可以了)：

Step1：编辑配置文件 Kconfig，加入驱动选项，使之在 make menuconfig 的时候出现

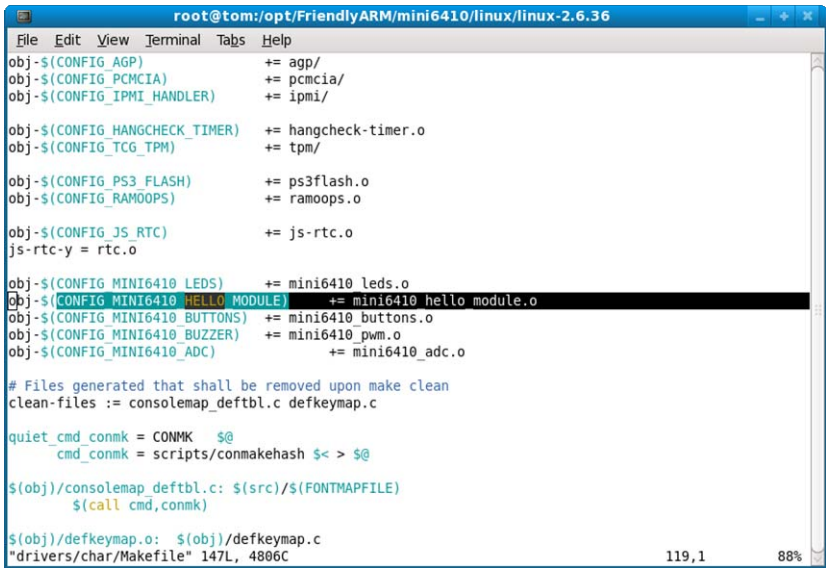
打开 linux-2.6.36/drivers/char/Kconfig 文件，添加如图所示：



保存退出，这时在 linux-2.6.36 目录位置运行一下 make menuconfig 就可以在 Device Drivers → Character devices 菜单中看到刚才所添加的选项了，按下空格键将会选择为<M>，此意为要把该选项编译为模块方式；再按下空格会变为<*>，意为要把该选项编译到内核中，在此我们选择<M>，如图，如果没有出现，请检查你是否已经装载了缺省的内核配置文件，见 1.5.1 章节。

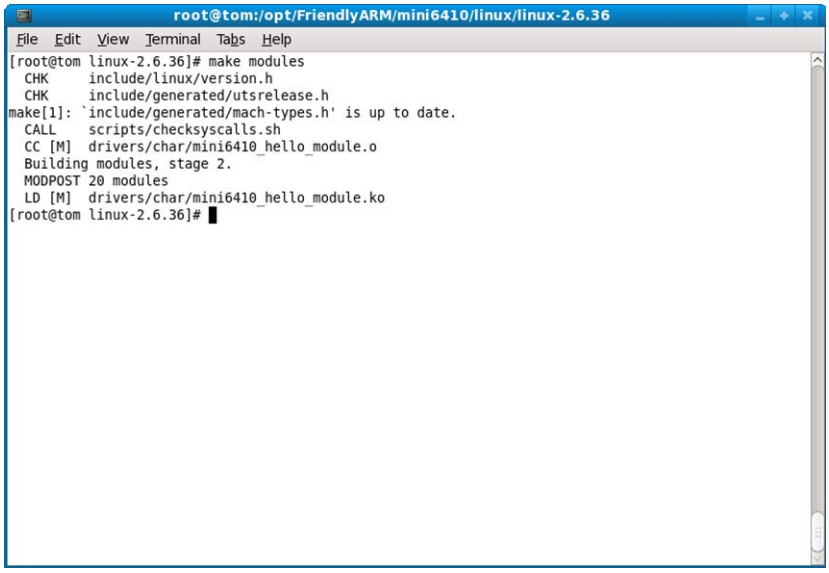


Step2：通过上一步，我们虽然可以在配置内核的时候进行选择，但实际上此时执行编译内核还是不能把 mini6410_hello_module.c 编译进去的，还需要在 Makefile 中把内核配置选项和真正的源代码联系起来，打开 linux-2.6.36/drivers/char/Makefile，如图添加并保存退出：



Step3：这时回到 linux-2.6.36 源代码根目录位置，执行 make modules，就可以生成我们所需要的内核模块文件 mini6410_hello_module.ko 了，注意：执行 make modules 之前，必须先执行 make zImage，只需一次就可以了。

至此，我们已经完成了模块驱动的编译。



(2)把 Hello, Module 下载到开发板并安装使用

在此使用 ftp 命令把编译出的 mini6410_hello_module.ko 下载到板子中，并把它移动到**/lib/modules/2.6.36-FriendlyARM** 目录然后在板子中现在执行

#modprobe mini6410_hello_module

可以看到该模块已经被装载了(注意：使用 modprobe 命令加载模块不需要加“ko”尾缀)

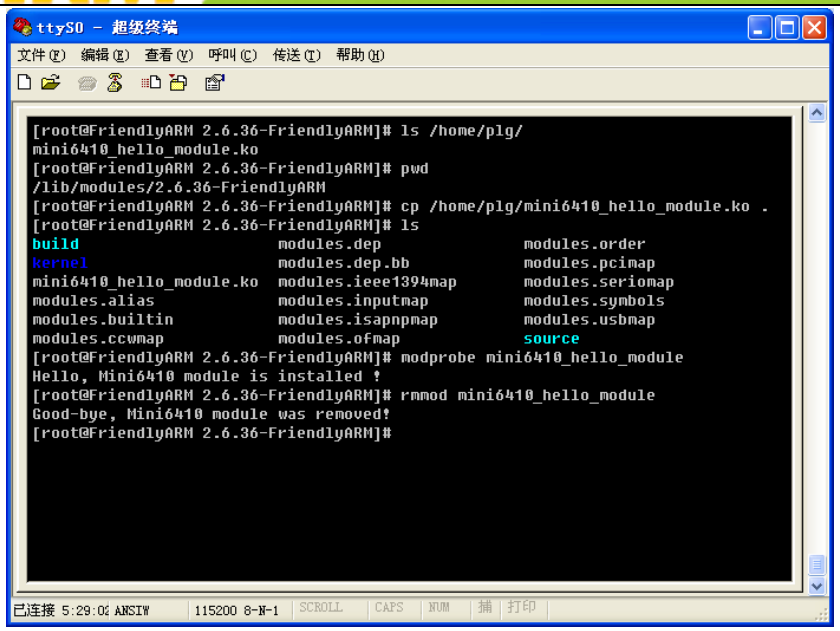
再执行以下命令，可以看到该模块被卸载

#rmmod mini6410_hello_module

注 意 ： 要 能 够 正 常 卸 载 模 块 ， 必 须 把 模 块 放 入 开 发 板 的 **/lib/modules/2.6.36-FriendlyARM** 目录

另外需要注意的是：因为我们的内核有时会升级更新，如果内核版本已经改变，请依照具体的内核版本重新建立一个模块存放目录，在此为**/lib/modules/2.6.36-FriendlyARM**

整个过程如下图：



1.10.2 LED 驱动程序

在上一小节，我们介绍了最简单的 Hello,Module 驱动程序模块，它只是从串口输出一些信息，并未对应板上的硬件进行操作，在嵌入式 Linux 系统中，大部分的硬件都需要类似的驱动才能操作，比如触摸屏、网卡、音频等，在这里我们介绍的是一些简单典型的例子，实际上复杂的驱动都有参考代码，不必从头写驱动。从本小节开始，我们将介绍一些和硬件密切相关的驱动，它们才是真正的嵌入式 Linux 驱动。

要写实际的驱动，就必须了解相关的硬件资源，比如用到的寄存器，物理地址，中断等，在这里，LED 是一个很简单的例子，它用到了如下硬件资源。

开发板上所用到的 4 个 LED 的硬件资源

LED	对应的 IO 寄存器名称	对应的 CPU 引脚
LED1	GPK4	R23
LED2	GPK5	R22
LED3	GPK6	R24
LED4	GPK7	R25

要操作所用到的 IO 口，就要设置它们所用到的寄存器，我们需要调用一些现成的函数或者宏，在此用到的是 readl 和 writel，它们将直接对相应的寄存器执行读取和写入的操作。在下面的驱动程序清单中，你可以看到它们被调用的情况。除此之外，你可能还需要调用一些和设备驱动密切相关的基本函数，如注册设备 misc_register，填写驱动函数结构 file_operations，以及像 Hello,Module 中那样的 module_init 和 module_exit 函数等。

有些函数并不是必须的，随着你对 Linux 驱动开发的进一步了解和阅读更多的代码，你自然明白。

程序源代码说明

驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char
驱动程序名称	Mini6410_leds.c
设备号	Led 属于 misc 设备，设备自动生成
设备名	/dev/leds
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/leds
测试程序名称	led.c
测试程序可执行文件名称	led
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

程序清单

```
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <asm/irq.h>
//#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/pci.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <asm/unistd.h>

#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>

#include <plat/gpio-cfg.h>
#include <mach/gpio-bank-e.h>
#include <mach/gpio-bank-k.h>
```

```
#define DEVICE_NAME "leds"

static long sbc2440_leds_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
	switch(cmd) {
		unsigned tmp;
	case 0:
	case 1:
		if (arg > 4) {
			return -EINVAL;
		}
		tmp = readl(S3C64XX_GPKDAT);
		tmp &= ~(1 << (4 + arg));
		tmp |= ( (!cmd) << (4 + arg) );
		writel(tmp, S3C64XX_GPKDAT);
		//printk (DEVICE_NAME": %d %d\n", arg, cmd);
		return 0;
	default:
		return -EINVAL;
	}
}

static struct file_operations dev_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl	= sbc2440_leds_ioctl,
};

static struct miscdevice misc = {
	.minor = MISC_DYNAMIC_MINOR,
	.name = DEVICE_NAME,
	.fops = &dev_fops,
};

static int __init dev_init(void)
{
	int ret;

	{
		unsigned tmp;
```

```
            tmp = readl(S3C64XX_GPKCON);
            tmp = (tmp & ~(0xffffU<<16))|(0x1111U<<16);
            writel(tmp, S3C64XX_GPKCON);

            tmp = readl(S3C64XX_GPKDAT);
            tmp |= (0xF << 4);
            writel(tmp, S3C64XX_GPKDAT);
        }

        ret = misc_register(&misc);

        printk (DEVICE_NAME"\tinitialized\n");

        return ret;
}

static void __exit dev_exit(void)
{
        misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

1.10.3 按键驱动程序

程序源代码说明		
驱动源代码所在目录	/opt/FriendlyARM/mini6410/linux/linux-2.6.36/drivers/char	
驱动程序名称	Mini6410_buttons.c	
该驱动的主设备号	Misc 设备，设备号将自动生成	
设备名	/dev/buttons	
测试程序源代码目录	/opt/FriendlyARM/mini6410/linux/examples/buttons	
测试程序源代码名称	buttons_test.c	
测试程序可执行文件名称	buttons	
说明：按键驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。		
开发板所用到的按键资源		
按键	对应的 IO 寄存器名称	对应的中断

K1	GPN0	EINT0
K2	GPN1	EINT1
K3	GPN2	EINT2
K4	GPN3	EINT3
K5	GPN4	EINT4
K6	GPN5	EINT5
K7	GPL11	EINT19
K8	GPL12	EINT20

程序清单

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <asm/irq.h>
#include <asm/io.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/hardware.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>

#include <plat/gpio-cfg.h>
#include <mach/gpio-bank-n.h>
#include <mach/gpio-bank-l.h>

#define DEVICE_NAME		"buttons"

struct button_irq_desc {
	int irq;
	int number;
	char *name;
};
```

```
static struct button_irq_desc button_irqs [] = {
	{IRQ_EINT( 0), 0, "KEY0"},
	{IRQ_EINT( 1), 1, "KEY1"},
	{IRQ_EINT( 2), 2, "KEY2"},
	{IRQ_EINT( 3), 3, "KEY3"},
	{IRQ_EINT( 4), 4, "KEY4"},
	{IRQ_EINT( 5), 5, "KEY5"},
	{IRQ_EINT(19), 6, "KEY6"},
	{IRQ_EINT(20), 7, "KEY7"},
};
static volatile char key_values [] = {'0', '0', '0', '0', '0', '0', '0', '0'};

static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

static volatile int ev_press = 0;


static irqreturn_t buttons_interrupt(int irq, void *dev_id)
{
	struct button_irq_desc *button_irqs = (struct button_irq_desc *)dev_id;
	int down;
	int number;
	unsigned tmp;

	udelay(0);
	number = button_irqs->number;
	switch(number) {
	case 0: case 1: case 2: case 3: case 4: case 5:
		tmp = readl(S3C64XX_GPNDAT);
		down = !(tmp & (1<<number));
		break;
	case 6: case 7:
		tmp = readl(S3C64XX_GPLDAT);
		down = !(tmp & (1 << (number + 5)));
		break;
	default:
		down = 0;
	}
```

```
	if (down != (key_values[number] & 1)) {
		key_values[number] = '0' + down;

			ev_press = 1;
			wake_up_interruptible(&button_waitq);
		}

	return IRQ_RETVAL(IRQ_HANDLED);
}


static int s3c64xx_buttons_open(struct inode *inode, struct file *file)
{
	int i;
	int err = 0;

	for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
	if (button_irqs[i].irq < 0) {
		continue;
	}
		err = request_irq(button_irqs[i].irq, buttons_interrupt, IRQ_TYPE_EDGE_BOTH,
					button_irqs[i].name, (void *)&button_irqs[i]);
		if (err)
			break;
	}

	if (err) {
		i--;
		for (; i >= 0; i--) {
		if (button_irqs[i].irq < 0) {
		continue;
		}
		disable_irq(button_irqs[i].irq);
			free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
		}
		return -EBUSY;
	}

	ev_press = 1;
```

```
	return 0;
}


static int s3c64xx_buttons_close(struct inode *inode, struct file *file)
{
	int i;

	for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
	if (button_irqs[i].irq < 0) {
		continue;
	}
	free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
	}

	return 0;
}


static int s3c64xx_buttons_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
{
	unsigned long err;

	if (!ev_press) {
	if (filp->f_flags & O_NONBLOCK)
		return -EAGAIN;
	else
		wait_event_interruptible(button_waitq, ev_press);
	}

	ev_press = 0;

	err = copy_to_user((void *)buff, (const void *)(&key_values), min(sizeof(key_values), count));

	return err ? -EFAULT : min(sizeof(key_values), count);
}

static unsigned int s3c64xx_buttons_poll( struct file *file, struct poll_table_struct *wait)
{
	unsigned int mask = 0;
```

```
	poll_wait(file, &button_waitq, wait);
	if (ev_press)
		mask |= POLLIN | POLLRDNORM;
	return mask;
}


static struct file_operations dev_fops = {
	.owner	=	THIS_MODULE,
	.open	=	s3c64xx_buttons_open,
	.release =	s3c64xx_buttons_close,
	.read	=	s3c64xx_buttons_read,
	.poll	=	s3c64xx_buttons_poll,
};

static struct miscdevice misc = {
	.minor = MISC_DYNAMIC_MINOR,
	.name = DEVICE_NAME,
	.fops = &dev_fops,
};

static int __init dev_init(void)
{
	int ret;

	ret = misc_register(&misc);

	printk (DEVICE_NAME"\tinitialized\n");

	return ret;
}

static void __exit dev_exit(void)
{
	misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("FriendlyARM Inc.");
```

1.11 编译 Qtopia-2.2.0

因为配置编译 Qtopia-2.2.0 的过程比较复杂，为了便于初学者学习和使用方便，我们把配置和编译的步骤制作成一个 **build** 脚本，只要执行该脚本即可编译整个 Qtopia 平台和应其自带的各个小程序；通过”**run**”脚本则可以运行它们，x86 和 arm 版本的步骤基本相同，而脚本内容则有稍微不同，下面是详细的步骤。

1.11.1 解压安装源代码

见 1.4.1 章节

1.11.2 编译和运行 x86 版本的 Qtopia-2.2.0

敬告：我们的软件开发和测试全部基于Fedora9 平台做开发，所有的配置和编译脚本也基于此平台，我们没有在其他平台上测试过。如果你对Linux开发很熟悉，相信你会根据错误提示逐步找到原因并解决，它们一般是你选用的平台缺少了某些库文件或者工具等原因造成的；否则，我们建议初学者使用和我们一致的平台，即 Fedora 9(全称为：Fedora-9-i386-DVD.iso)，你可以在其官方网站下载(ftp://download.fedora.redhat.com/pub/fedora/linux/releases/9/Fedora/i386/iso/Fedora-9-i386-DVD.iso，不保证长期有效)，也可以在其他地方获取，它们都是一样的，安装时请务必参考我们手册提供的步骤，这是我们经过严格测试的，以免遗漏一些开发时所需要的组件。

Linux 的发行版本众多，我们无法为此一一编写文档解释安装方法，请谅解。

进入工作目录，执行以下命令

#cd /opt/FriendlyARM/mini6410/linux/x86-qtopia

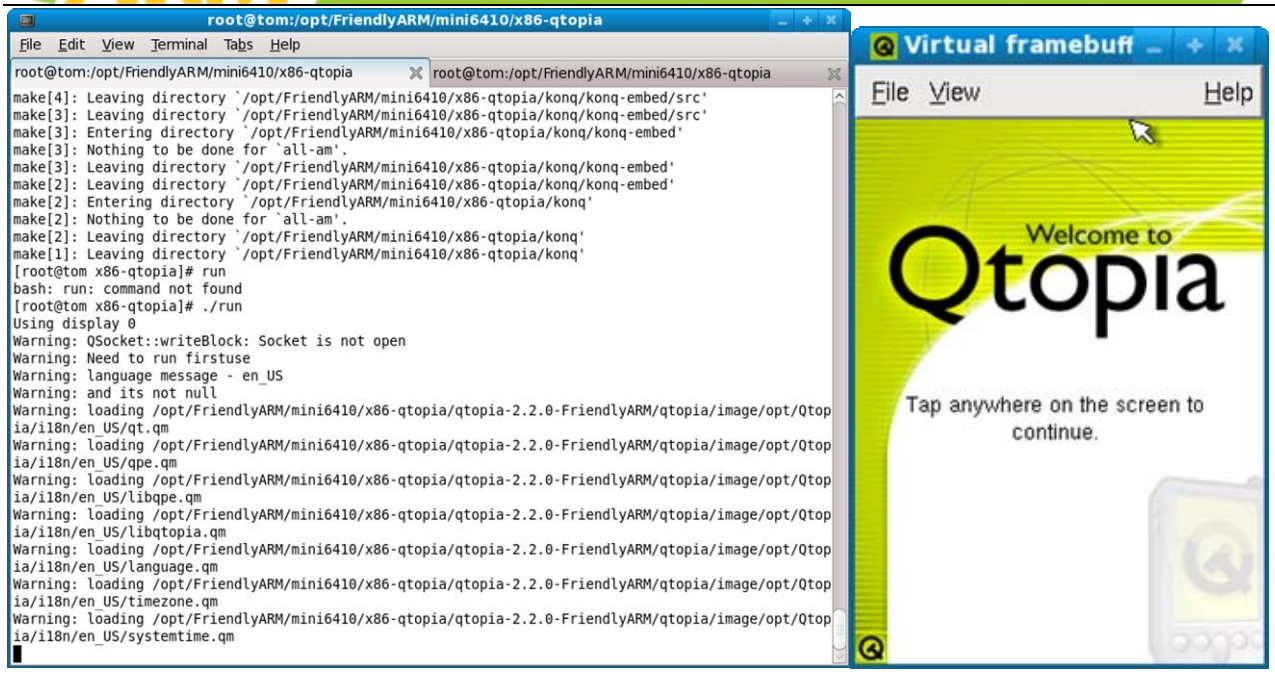
#./build-all　　(该过程比较长，需要运行大概 30 分钟左右)

说明：./build-all 将自动编译完整的 Qtopia 和嵌入式浏览器，你还可以先后执行**./build** 和**./build-konq** 脚本命令分别编译它们，如果你没有顺利执行，请参考上面的红色字体说明。

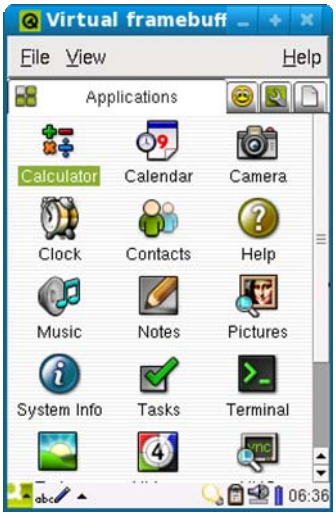
要运行你刚刚编译出的 Qtopia 系统十分简单，在刚刚编译完的命令终端下输入如下命令：

#./run ；注意，“/”前面有个“.”，这表示在当前目录执行

这时你可以看到如下界面



按照提示点击运行就可以看到 Qtopia 系统了，如下，
注意：我们并没有制作 x86 版本的中文系统。



1.11.3 编译和运行 arm 版本的 Qtopia-2.2.0

请确认你所使用的编译器版本为 arm-linux-gcc-4.4.1，运行平台为 Fedora 9，进入工作目录，执行以下命令

#cd /opt/FriendlyARM/mini6410/linux/arm-qtopia

#./build-all　　(该过程比较长，需要运行大概 30 分钟左右)

#./mktarget　　(制作适用于根文件系统的目标板二进制映象文件包，将生成 **target-qtopia-konq.tgz**)

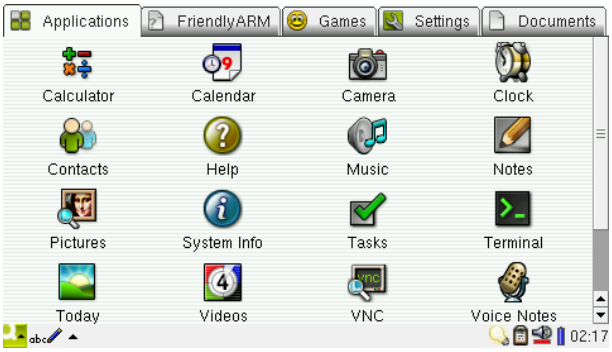
说明：./build-all 将自动编译完整的 Qtopia 和嵌入式浏览器，并且编译生成的系统支持 Jpeg、GIF、PNG 等格式的图片，您还可以先后执行**./build** 和**./build-konq** 脚本命令分别编译它们。如果你没有顺利执行，请参考上一节开头的红色字体说明。

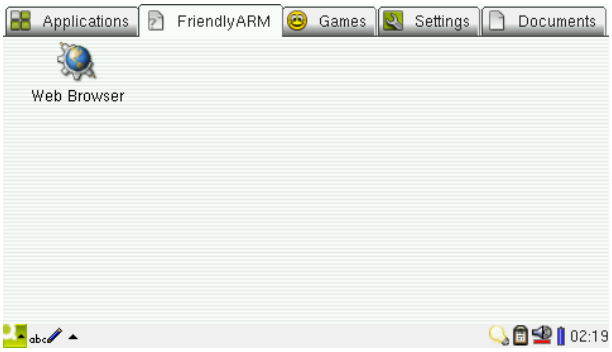
你可以删除开发板中原有的 Qtopia 系统，只要把/opt 目录下的所有文件都删除就可以了。然后把刚刚生成的 target-qtopia-konq.tgz 通过优盘或者其他方式解压到开发板的根目录，假定我们已经通过 ftp 把它传到了/home/plg 目录下，然后在开发板命令终端执行：

#tar xvzf /home/plg/target-qtopia-konq.tgz –C /

其中"C"是 Change 的意思，"C"后面的"/"代表要解压到根目录下，执行完毕，重启开发板，就可以看到所有的界面都已经变为英文的，并且"FriendlyARM"标签下只有一个浏览器程序，这就是你自己编译得到的整个 Qtopia 系统了，如下图：

注意：新系统可能会使用预装系统的触摸屏校正参数/etc/pointercal，你也可以在删除旧系统的时候，一并删除它，这样开机后就会进入校正界面了。





上面的过程看似很简单，其实所有的秘密都在 build-all 脚本中，网上也有很多关于移植的文章，但本质的步骤都是脚本所记录的那些，你可以使用记事本工具打开自行查看一下，只不过其中没有那些嘻哈打诨的辞藻罢了。

1.12 编译 QtE-4.7.0

1.12.1 解压安装源代码

见 1.4.1 章节

1.12.2 编译和运行 arm 版本的 QtE-4.7.0

注意，请务必使用我们光盘中提供的交叉编译器 arm-linux-gcc-4.5.1，并基于 Fedora9 平台，Fedora9 的安装步骤见 1.3.1，我们不能保证所制作的脚本可以在其他平台环境下顺利执行，对此我们也不提供支持。

和 Qtopia-2.2.0 十分类似，我们也为 QtE-4.7.0 的编译制作了现成的脚本 build-all，进入源代码目录执行：

#cd /opt/FriendlyARM/mini6410/linux/arm-qte-4.7.0

#./build-all

这个过程将十分漫长，根据机器配置不同，会有不同的编译时间，请耐心等待。

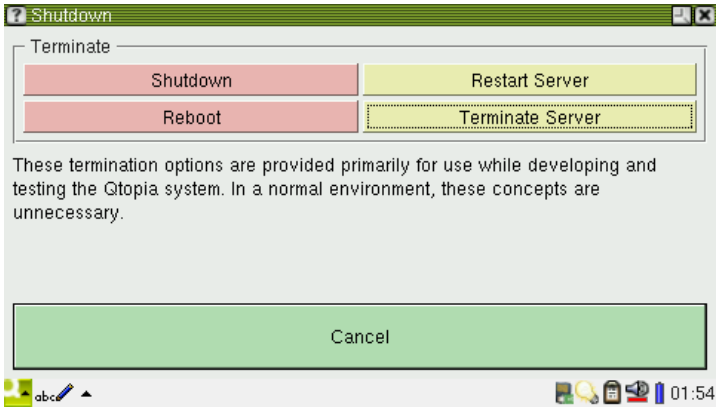
当顺利执行完毕，再运行 mktarget 脚本，将会从编译好的目标文件目录中，提取出必要的 QtE-4.7.0 库文件和可执行二进制示例，并打包为 **target-qte-4.7.0.tgz**，把它在开发板的根目录下解压，就可以使用了，如下命令

#tar xvzf target-qte-4.7.0.tgz –C /

这样，就会在/usr/local/目录下创建生成 Trolltech 目录，它里面包含了运行所需要的所有库文件和可执行程序。

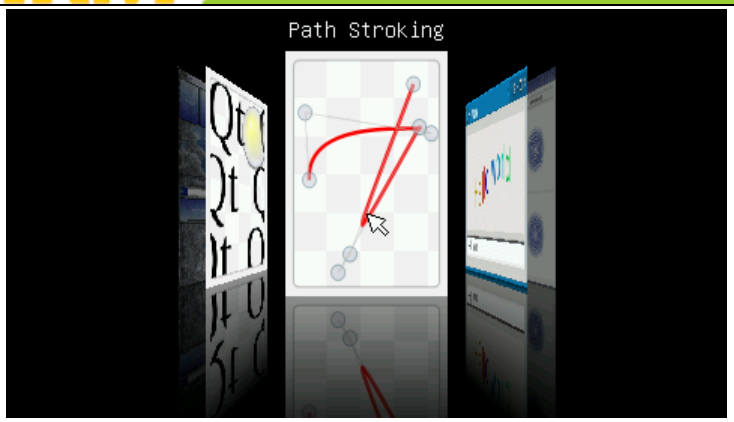
使用说明：因为目标板中预装的 Linux 已经包含了 QtE-4.7.0，测试之前，你可以先删除原有的，它位于**/usr/local/Trolltech** 目录，把该目录使用 **rm** 命令完全删除就可以了。

要运行 QtE-4..6.3，建议先停止正在运行的 Qtopia-2.2.0，点“设置”中的“关机”可出现如下界面，点“Terminate Server”即可关闭 Qtopia-2.2.0 系统。



也可以使用其他的方法，比如在启动脚本/etc/init.d/rcS 中注释掉 qtopia 启动项，再重新系统系统；或者使用 killall 命令杀死相关的进程(比较多；甚至是直接删除/opt 目录中的所有内容重启。

关闭 qtopia-2.2.0 之后，在命令行输入 qt4 命令，即可启动刚刚解压安装的 QtE-4.7.0 了，如图：



1.13 编译 Qtopia4(Qt-Extended-4.4.3)

1.13.1 解压安装源代码

见 1.4.1 章节

1.13.2 编译和运行 x86 版本的 Qt-Extended-4.4.3

注意，请务必使用我们光盘中提供的交叉编译器 arm-linux-gcc-4.5.1，并基于 Fedora9 平台，Fedora9 的安装步骤见 1.3.1，我们不能保证所制作的脚本可以在其他平台环境下顺利执行，对此我们也不提供支持。

和 Qtopia-2.2.0 十分类似，我们也为 Qt-Extended-4.4.3(以下简称 Qtopia4)的编译制作了现成的脚本 build，进入源代码目录执行：

#cd /opt/FriendlyARM/mini6410/linux/ x86-qt-extended-4.4.3

#./build

这个过程将十分漫长，根据机器配置不同，会有不同的编译时间，请耐心等待。

要运行你刚刚编译出的 Qtopia4 系统十分简单，在刚刚编译完的命令终端下输入如下命令：

#./run ；注意，“/”前面有个“.”，这表示在当前目录执行

这时你可以看到如下界面，依提示点进去可以看到功能表格，如果你有兴趣，可以自行点击看看里面的功能，在此不再赘述。

1.13.3 编译和运行 arm 版本的 Qt-Extended-4.4.3

注意，请务必使用我们光盘中提供的交叉编译器 arm-linux-gcc-4.5.1，并基于 Fedora9 平台，Fedora9 的安装步骤见 1.3.1，我们不能保证所制作的脚本可以在其他平台环境下顺利执行，对此我们也不提供支持。

和 Qtopia-2.2.0 十分类似，我们也为 Qt-Extended-4.4.3 的编译制作了现成的脚本 build，进入源代码目录执行：

#cd /opt/FriendlyARM/mini6410/linux/ arm-qt-extended-4.4.3

#./build

这个过程将十分漫长，根据机器配置不同，会有不同的编译时间，请耐心等待。

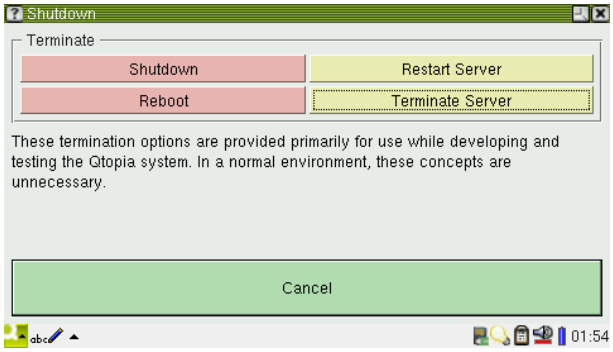
当顺利执行完毕，再运行 mktarget 脚本，将会从编译好的目标文件目录中，提取出必要的 **Qt-Extended-4.4.3** 库文件和可执行二进制示例，并打包为 **target-qtopia4.tgz**，把它在开发板的根目录下解压，就可以使用了，如下命令

#tar xvzf target-qtopia4.tgz –C /

这样，就会在/opt 目录下创建生成 Qtopia4.4.3 目录，它里面包含了运行所需要的所有库文件和可执行程序。

使用说明：因为目标板中预装的 Linux 已经包含了 QtE-4.7.0，测试之前，你可以先删除原有的，它位于**/opt/** Qtopia4.4.3 目录，把该目录使用 **rm** 命令完全删除就可以了。

要运行 Qtopia4，建议先停止正在运行的 Qtopia-2.2.0，点“设置”中的“关机”可出现如下界面，点“Terminate Server”即可关闭 Qtopia-2.2.0 系统。



也可以使用其他的方法，比如在启动脚本/etc/init.d/rcS 中注释掉 qtopia 启动项，再重新系统系统；或者使用 killall 命令杀死相关的进程(比较多；甚至是直接删除/opt 目录中的所有内容重启。

关闭 qtopia-2.2.0 之后，在命令行输入“qtopia4 &”(这里的“&”表示后台运行)命令，即可启动刚刚解压安装的 Qtopia4 了，如图：

1.14 选择哪个版本的 Qt 进行开发

面对这么多版本的 Qt，有很多用户不知道该如何选择，其实这也没有标准的答案，我们认为这主要取决于你的需求而定，并不是最新的就是好的，也不是老版本的就不好。

对于开发板平台而言，我们需要一套完整的桌面系统(Qtopia 就是手持设备的桌面系统)，以便适合于在各种分辨率的 LCD 上都可以有不错的显示效果，用于展示开发板的各项功能，因此我们基于 Qtopia-2.2.0 开发了一些小程序，并实现了和 Qtopia4，QtE-4.7.0 等的共存和自由切换，这些实现实质上并没有采用很新很时尚的技术，它们都是比较基本的 Linux C 或 C++编程，图形界面只不过是个外壳，但这已经达到了我们的需求。

如果你的应用不需要整套的桌面系统，只是单独的一两个应用程序，我们推荐使用 QtE-4.7 或更高版本，因为它们的跨平台开发性更好，对于初学者可能更容易掌握和移植，需要说明的是，单独的 QtE-4.7 应用程序占用的空间并不是很大。