愿做您成功路上的基石！

——君益兴

HELPER2416_V2 开发板

LINUX 用户手册

V1.11

2016-10-17

百杰科技 BAIJIE TECHNOLOGY

www.jyxtec.com

深圳市百杰科技有限公司

地址：深圳市宝安区西乡街道宝源路宝安互联网产业基地 A 区 1 栋 202B2 电话：0755-86017779

修订记录

日期	版本	修订说明	作者
2014.02.24	1.0	新建	陈军
2014.02.25	1.1	增加 128M 内存时的 UBOOT 编译描述	陈军
2014.02.28	1.2	修改 tar 解压描述	陈军
2014.03.03	1.3	增加编译内核之前打补丁的描述	陈军
2014.03.05	1.4	去除关于声卡工作时 USB 不正常的描述，WM8731 没有该问题	陈军
2014.03.31	1.5	修改制作内核 logo 的描述	陈军
2014.04.04	1.6	修改 SD 卡烧写描述	陈军
2014.04.17	1.7	增加不同频率，不同内存的对应 uboot 描述	陈军
2014.07.05	1.8	修改 LED 操作描述（感谢 mensy 提出 BUG）	陈军
2014.08.17	1.9	修改 nfs 启动的描述	陈军
2014.10.14	1.10	修改制作根文件系统的描述	陈军
2016.10.17	1.10	修改部分描述，更改公司名	陈军

目录

修订记录 2
目录 3
1 Helper2416 开发板介绍 6
1.1 简介 6
1.2 系统特性 6
1.3 开发板硬件资源描述 7
2 开发板功能测试 9
2.1 开发板的设置及连接 9
2.1.1 启动模式选择 9
2.1.2 外部接口连接 9
2.1.3 安装 USB 转串口驱动 10
2.1.4 超级终端 10
2.2 硬件测试 12
2.2.1 开机测试 12
2.2.2 JTAG 测试 13
2.2.3 SD 卡接口测试 15
2.2.4 键盘测试 16
2.2.5 可编程 LED 测试 16
2.2.6 串口测试 16
2.2.7 网口测试 17
2.2.8 声卡测试 18
2.2.9 USB Host 测试 20
2.2.10 USB Device 测试 20
2.2.11 E2PROM 测试 20
2.2.12 CAN 总线测试 20
3 开发板平台构建 21
3.1 使用 vmware 虚拟机镜像构建虚拟机开发环境 21
3.2 烧写 LINUX 系统到板载 NAND Flash 30
3.2.1 烧写系统镜像到 TF 卡 30
3.2.2 从 SD 烧写 LINUX 系统到 NAND Flash 32
3.3 用 tftp 搭建板子系统 33
3.3.1 启动 tftp 服务 33
3.3.2 设置 u-boot 环境变量 35
3.3.3 烧写 u-boot 35
3.3.4 烧写 kernel 36
3.3.5 烧写根文件系统 36

4 源码编译 38
4.1 编译 u-boot 38
4.2 编译内核 39
4.2.1 编译内核版本 3.2 39
4.2.2 特别说明 39
4.3 编译模块 40
4.4 修改开机画面 40
4.5 制作根文件系统 40
4.6 编译 Qtopia2.2 41
4.6.1 交叉编译 jpeg 42
4.6.2 交叉编译 e2fsprogs 42
4.6.3 交叉编译 png 42
4.6.4 交叉编译 zlib 43
4.6.5 交叉编译 tslib 43
4.6.6 交叉编译 qtopia 44
4.7 编译 qt4 46
4.8 编写 Hello World 程序 46
5 传输文件到目标板 48
5.1 使用 U 盘传输文件 48
5.2 使用 SD 卡传输文件 48
5.3 使用串口传输文件 48
5.4 使用 usb-gadget 方式传输文件 50
6 网络服务 51
6.1 配置开发板网络接口 51
6.2 telnet 52
6.3 ftp 53
6.4 http 54
6.5 nfs 54
7 内核配置 56
7.1 网卡驱动 56
7.2 MMC 卡驱动 57
7.3 USB 驱动 58
7.4 LCD 驱动 60
7.5 触摸屏驱动 61
7.6 声卡驱动 63
7.7 文件系统配置 65
7.8 usb-gadget 配置(将开发板当 U 盘使) 71
8 附录 73

8.1 在 VirtualBox 虚拟机里安装 linux（ ububtu 10.10 ）.....73
8.1.1 配置虚拟机网络连接.....86
8.1.2 光驱设置.....87
8.1.3 主机与虚拟机间共享文件.....88
8.1.4 安装开发环境.....89
8.1.5 安装交叉编译器.....90
8.2 注意事项.....90
8.3 编译小技巧.....91
8.4 触摸屏校准.....91
8.5 关于触摸屏与鼠标共存的问题.....92
8.6 一个 LINUX 应用程序例子--抓屏软件源码.....93
8.7 开发一个 QTOPIA 应用程序--目标板 LED 控制.....95
8.8 关于部署 LINUX 开发环境的建议.....102
8.9 推荐几个好网站：.....102
8.10 存在问题.....**错误!未定义书签。**
8.11 关于技术支持.....102
8.12 关于定制服务.....103

1 Helper2416 开发板介绍

1.1 简介

Helper2416，得名于我曾经的一个软件作品——EDAHelper，即 EDA 助手（硬件工程师的设计辅助工具），我希望 Helper2416 能成为嵌入式初学者学习的助手。

Helper2416 开发板，是一款低价高性能的 ARM926EJ-S 学习评估板，由本公司自主研发、生产及其销售。采用三星的 S3C2416 作为主处理器，板载丰富的资源及接口，配套完善的 LINUX 驱动及应用软件，是个人学习嵌入式 LINUX 开发及企业前期产品评估的理想选择。本手册在没有特别指明的情况下，默认采用 4.3 寸液晶屏。

当前 Helper2416 已经升级到第二版，采用核心板+底板结构，结合用户的需求和我们自己的项目积累，第二版与第一版相比，引出了大量 GPIO 口，增加了 RTC 电池座，Mini-USB 换成了智能手机充电线通用的 MicroUSB，接口设计更合理，重要接口都增加了接口保护，布局更漂亮，成本也做是很大的优化，总之，性能提高了，价格更低了。

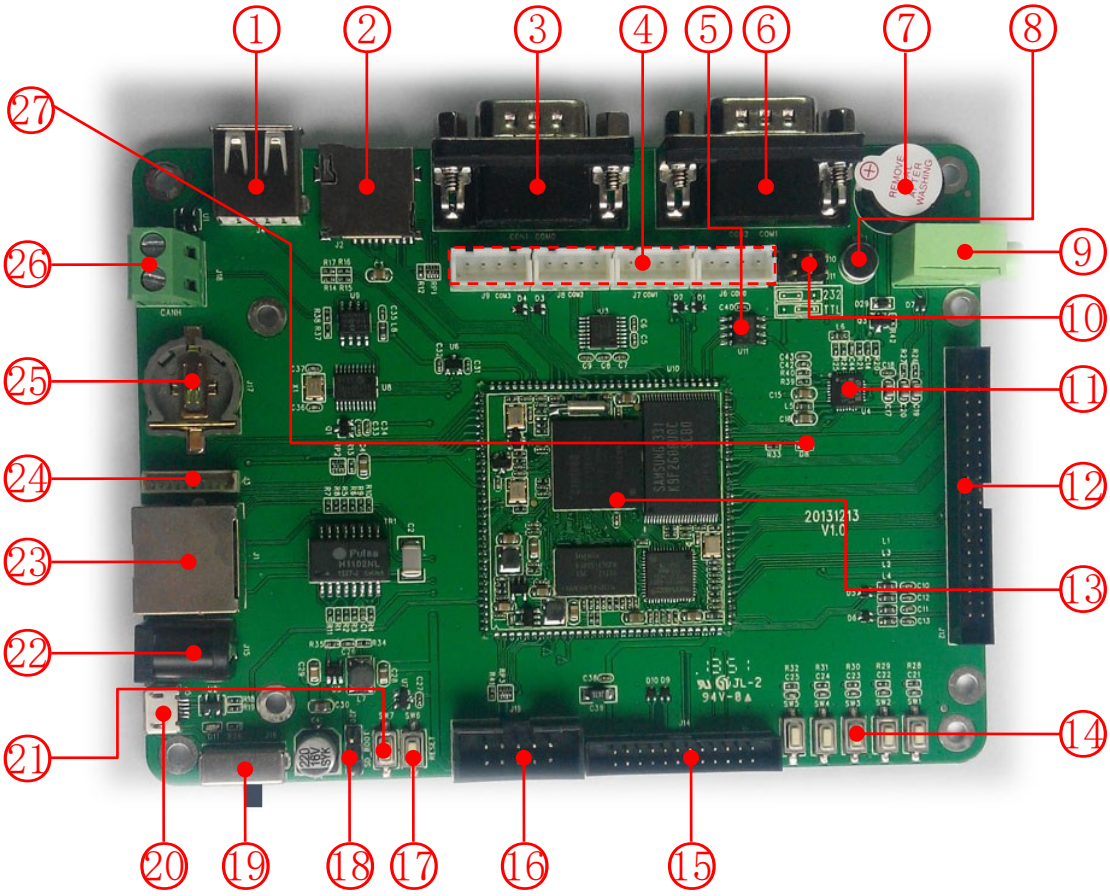
限于水平，本手册难免会有疏漏和错误，欢迎批评指正，不甚感激。

1.2 系统特性

- 400/533MHz 主频 CPU 稳定运行
- ARM926EJ-S 内核，支持 JAVA 加速
- 256M SLC Nand Flash
- 64M DDR2/128M mDDR(可选) SDRAM 300MHz 时钟稳定运行
- 16Kbit I2C EEPROM
- 10 针 5x2-2.54mm JTAG 接口，送转接板
- 支持 1 路声卡
- 支持 2D 图形加速
- 支持 1 路 SD 接口存储卡，1 路 SDIO 引出插座
- 支持 SD 卡（绝大多数 SD 卡）启动、Nand Flash 启动
- 独创一键选择 SD 卡 / NAND 启动
- 支持 2 路高速 RS232 串口(三线)或者 4 路 TTL 串口（与 RS232 串口复用，跳线选择，带接口保护）
- 支持 1 路 CAN 总线
- 支持 1 路 USB Host，1 路 USB Device
- 支持 4.3 寸、5.6 寸、7 寸、8 寸、10.2 寸等液晶屏，最高分辨率 1024x768@30Hz，带触摸
- 支持 WIFI、GPS 等外置模块
- 支持 10/100BASE-TX 以太网，收发自动翻转

- 支持 1 个可编程 LED 灯，1 个电源灯
- 支持 5 个普通按键、1 个复位按键
- 支持 USB 及外部电源供电
- 5V 电源供电（4.3 寸及 5.6 寸液晶屏可直接用 USB 供电）
- 外扩接口包含 9 个中断 IO 口，2 个 PWM，4 个 ADC，5 个普通 IO 口，3.3V 电源输出(100mA)
- 支持 usb-gadget，开发板上的 NAND 可以当成普通的 U 盘连接到 PC 机

1.3 开发板硬件资源描述



位置	描述
①	USBA 型 Host 接口
②	SD1，TF 卡插座，该 TF 卡接口支持内核动态检测卡拔插，支持 TF 卡启动，由于 TF 卡也是一种 SD 卡，以下描述统称 SD 卡
③	RS232-COM0，u-boot 及 LINUX 系统打印信息出入口
④	4 个 TTL COM 口，其中 COM0，COM1 与 RS232 复用，跳线⑩选择

⑤	AT24C16，EEPROM，16Kbit，I2C 接口
⑥	RS232-COM1
⑦	蜂鸣器
⑧	板载麦克风
⑨	耳机输出口
⑩	COM0/COM1 的 TTL / RS232 选择跳线
⑪	WM8731 声卡
⑫	40 针液晶屏接口，含电阻式触摸屏接口（引脚定义参见原理图）
⑬	HELPER2416 核心板
⑭	键盘
⑮	24 针 IO 扩展口
⑯	10 针 JTAG 接口，配转接板（引脚定义参见原理图）
⑰	复位按键
⑱	SD / NAND 启动选择，该位置可以配置为始终从 SD 卡启动
⑲	电源开关
⑳	MicroUSB 接口，兼容智能手机充电线，可直接用充电线供电，可配置为 USB Host
㉑	SD / NAND 启动选择，一键触发
㉒	5V 电源插座
㉓	RJ45 以太网接口
㉔	引出 SDIO0，可用于扩展 SD 卡、SDIO WIFI 等，该接口不支持从 SD 卡启动
㉕	CR1220 电源插座，本开发板不配电池，但保证其能正常工作
㉖	CAN 总线接口

2 开发板功能测试

开发板出厂时默认烧写的是 LINUX 系统，对应软件包里 Helper2416/image/u-boot.bin，zImage， root-qtopia.img，不同型号的液晶屏需要烧写对应的内核（如 4.3 寸液晶屏烧写 zImage.43）。拿到板子即可上电进行硬件系统测试，以便确保开发板各功能部件工作正常。

2.1 开发板的设置及连接

2.1.1 启动模式选择

本开发板支持 TF 卡启动（注意：S3C2416 只有 SD1 那个接口可以支持 SD 卡启动，1.3 节图中②）和 NAND 启动，通过 1.3 节图中的跳线⑱或者按键㉑来选择，当跳线⑱配置为  时是从 SD 卡启动，当跳线⑱配置为  时，跳线⑱无作用（此为开发板默认配置），此时可以按住 SW7 不放，再按复位键 SW6，并松开，可从 TF 卡启动，当然，前提是你已经有可以启动的 TF 卡插入 TF 卡座中，除此之外，都为 NAND 启动。

2.1.2 外部接口连接

电源：使用本公司提供的 5V 电源适器连接电源插座（1.3 节图中㉒）或者使用本公司提供的 USB 线连接 MicroUSB 接口（1.3 节图中㉑）

注意：如果使用自有电源适配器，最好是 5V/2A 的，本身开发板功耗只有几百毫安，但是很多市场上的电源适配器在输出标称电流时，压降很大，小于 2A 的适配器可能带不动液晶屏；另外，本公司提供的 USB 线是质量非常好，5.6 寸以下的液晶屏可以用该 USB 线直接供电。

USB Device：直接用本公司提供的 USB 线连接开发板的 MicroUSB 口和 PC 机的 USB 口，通过该方式可以将 PC 机上的文件直接拷贝到开发板的存贮设备中，如 SD 卡等

USB Host：可直接将 U 盘、键盘、鼠标等设备直接插入开发板的 USB Host 接口（1.3 节图中①），内核已经支持这些设备

串口：使用交叉双母头串口线连接开发板的 COM0（1.3 节图中③）和 PC 机的串口

网口：使用普通网线连接开发板的网口（1.3 节图中㉓）与交换机或者 PC 机网口（本开发板的网卡支持 HP-Auto MDIX，交叉、直连线都 OK）

LCD 接口：使用本公司配套的 40 针 2.0mm 的排线连接 LCD 接口（1.3 节图中⑫）及液晶屏驱动板

JTAG：使用本公司提供的 JTAG 转接板和排线连接普通的 JTAG 仿真器

耳机：WM8731 能自动检测耳机插入，以决定输出到外响或者耳机，普通的 3.5mm 耳机插入耳机座即可（1.3 节图中⑨）

TF 卡槽：普通标准的 TF 卡，最大支持 32G，同样支持 SDIO 接口的 WIFI 模块等

2.1.3 安装 USB 转串口驱动

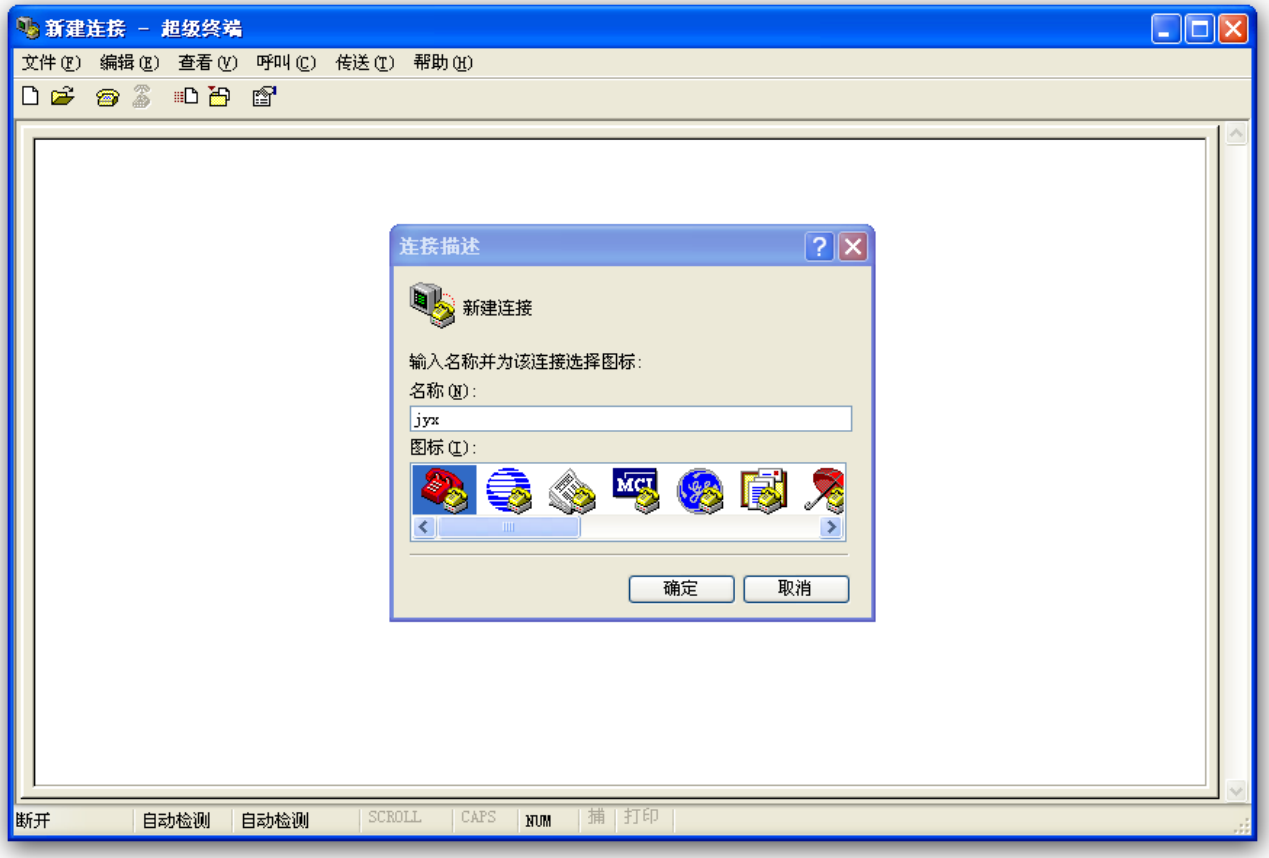
如果电脑没有串口，则可以购买 USB 转串口线，如果购买了本公司提供的 USB 转串口头，请安装软件包里提供的驱动：Helper2416\tools\USB 转串口驱动\PL2303\。

2.1.4 超级终端

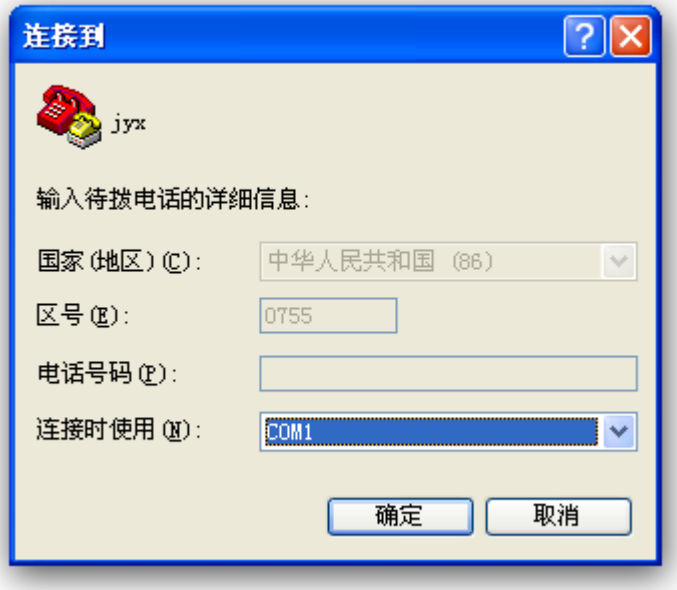
对于嵌入式开发而言，大量的调试信息都是从串口打印，我们需要一个串口调试软件，以监控开发板的运行状态。

常用的串口调试软件有 sscom32，SecureCRT，超级中端等。下面介绍超级终端的使用。

如果您的系统是 Win7， 在 Win7 下默认没有带超级终端，可使用本公司提供的软件包里：Helper2416/tools/超级终端，直接运行 hypertrm.exe；如果您的系统是 XP，则可从开始菜单进入：开始->所有程序->附件->通讯->超级终端，如下图：



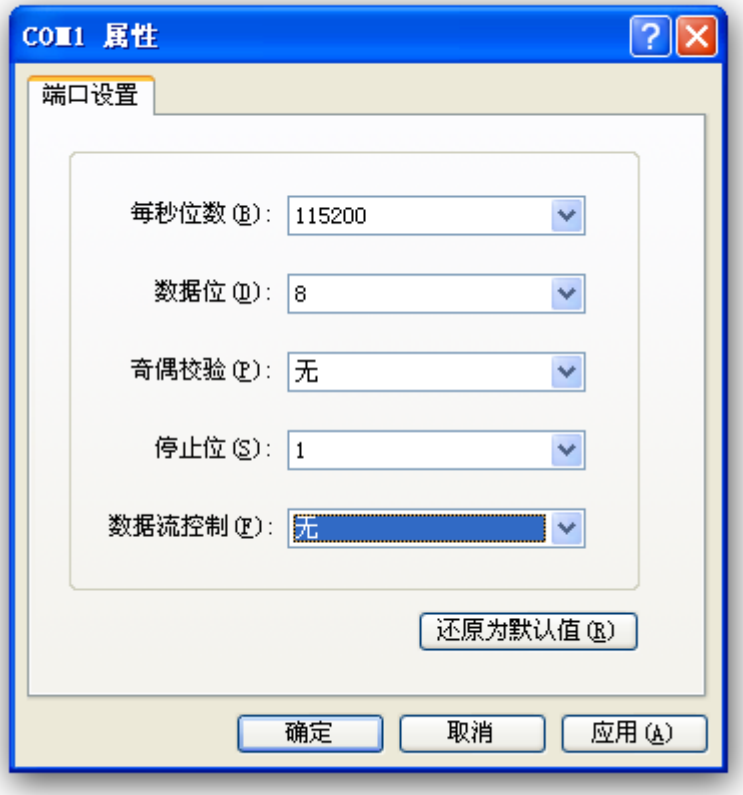
输入任意名称->确定，进入如下设置窗口：



选中与开发板连接的串口，再确定，如果在“连接时使用（N）“处，没有 COM 选项，请确认 PC 机是否有串口设备，如果没有，可以使用 USB 转串口线。

如下图，修改每秒位数（波特率）115200，数据位 8 位，无奇偶校验，1 位停止位，

数据流控制选择 无，然后点击确定。



然后点击左上角菜单 文件->保存，以后再使用超级中端，可以直接从文件菜单打开，无需重新设置。

2.2 硬件测试

以下测试全部是在安装有本公司提供的 LINUX 系统的开发板硬件上进行。全部测试通过则代表硬件合格。

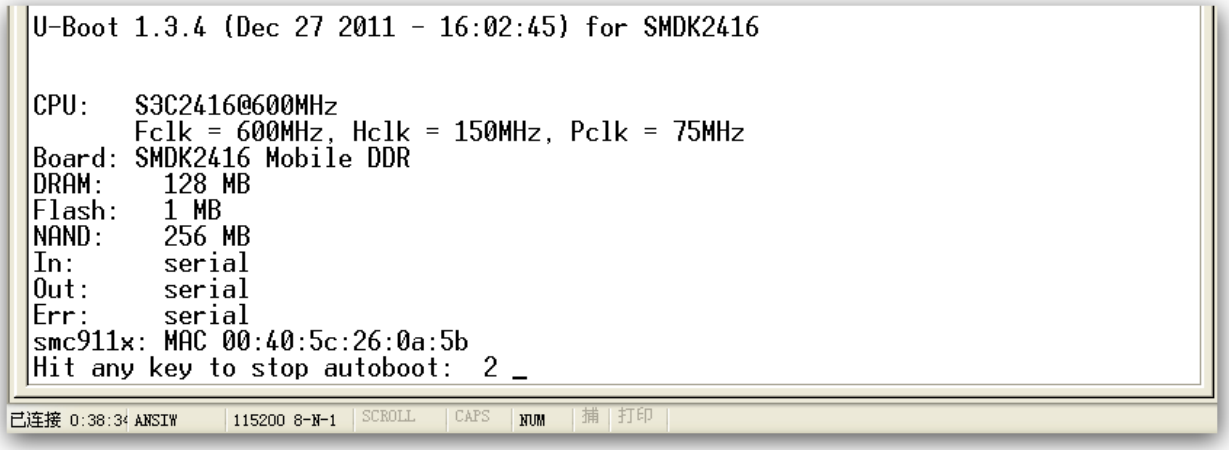
2.2.1 开机测试

开发板默认安装 LINUX 系统软件，可以直接上电运行看效果。

先打开并设置好超级终端（参考 2.1.4 节）。再参考 2.1.2 节，按顺序接通串口 0、LCD 驱动板连线、网线（如果需要的话）、电源、USB 线。

注意：请把开发板放置在平整桌面上，不要垫高，特别是在插 LCD 座的时候一定要小心。

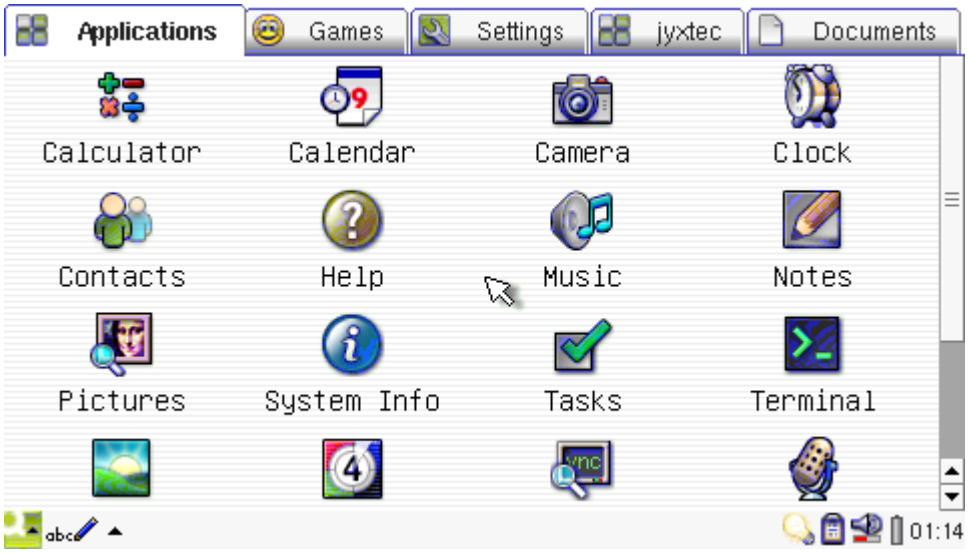
最后打开电源开关（1.3 节图中⑲），超级终端里打印 u-boot 提示信息：



u-boot 等待 3 秒（实际产品中无需等待，请自行设置启动等待时间）后启动 LINUX 系统。液晶屏上出现开机画面，如下图：

再等待 20 秒左右，进入触摸屏校准，用笔状物，也可用手指甲依次点击屏幕上的“十“字即可。如果不小心点错了地方，可以参考 8.3 节，重新进入校准。第一次启动会校准触摸屏，校准数据会保存到 Flash 中，此后再次启动就不会进入触摸屏校准了。

最后进入 qtopia 桌面就表示启动完成了。如下图：



到此为止，已经表明液晶屏和触摸屏已经工作正常了。

2.2.2 JTAG 测试

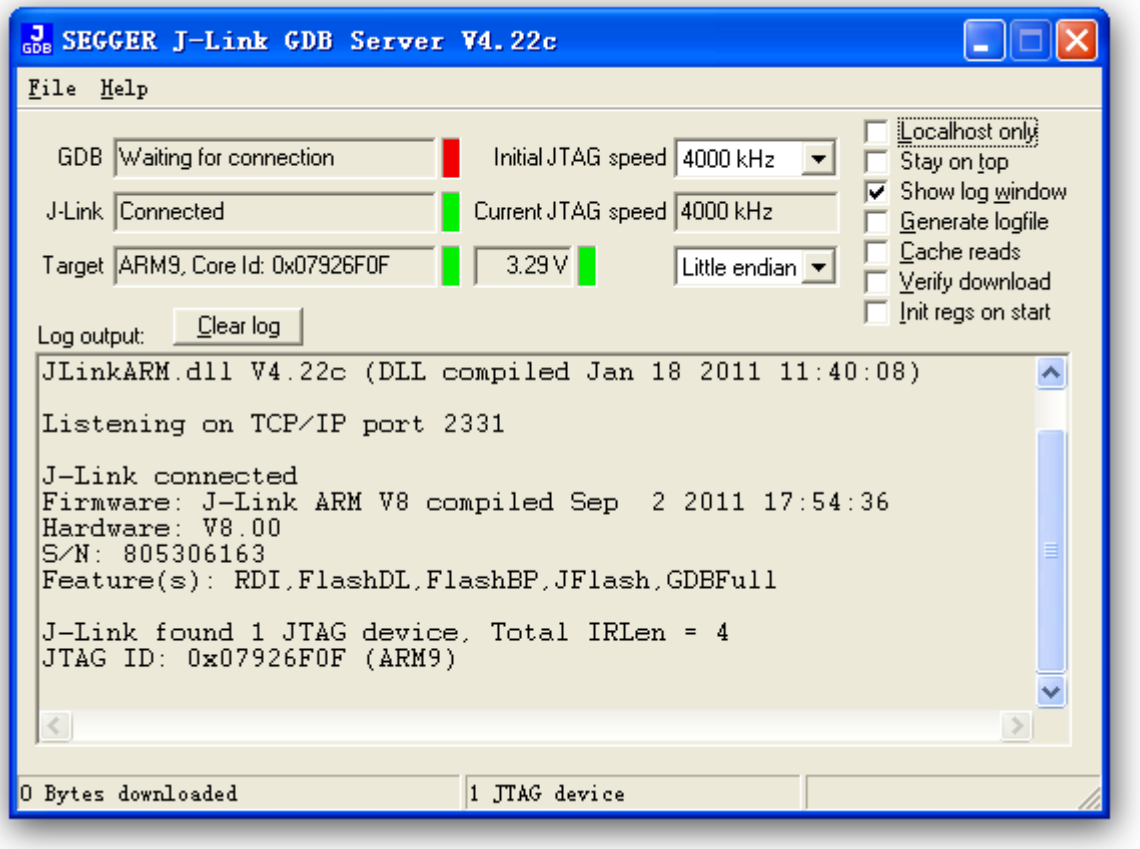
如果有购买本公司提供的 JTAG 仿真器，可以通过转接板和排线连接仿真器到开发板。手里没有 JTAG 仿真器则忽略本节。当然，连接 JTAG 的时候要先关电源。

2.2.2.1 J-Link v8 测试

如果您购买的是 J-Link v8 仿真器，请自行下载并安装 J-Link 支持软件，下载地址：

http://www.segger.com/jlink-software.html

序列号随便输，安装好后，运行 segger 软件里 j-link GDB server，开始->程序->SEGGER->J-Link ARM V4.22c->J-Link GDB Server，如下图：



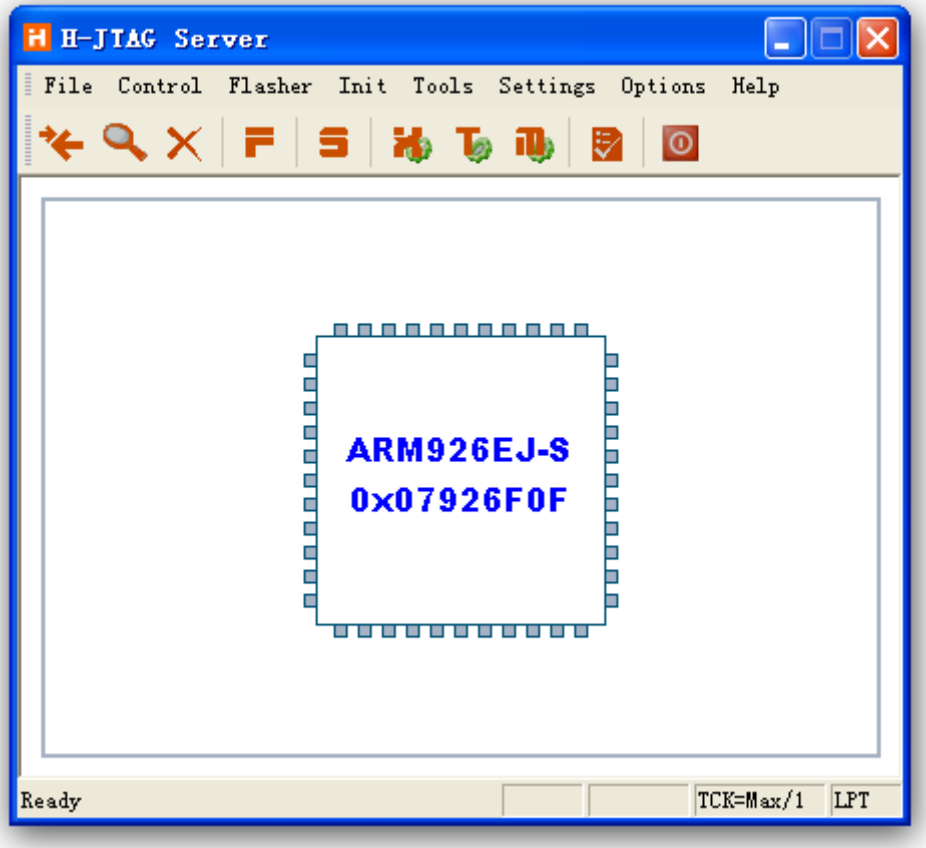
如果上图 Target 区域信息与您看到的信息一致，则说明仿真器识别到了 CPU，JTAG 接口工作正常。

2.2.2.2 Wiggler-JTAG 测试

如果您购买的是 Wiggler-JTAG，同样可以检测 JTAG 是否正常。请自行下载并安装 HJTAG（一个优秀的国产 JTAG 仿真服务器软件），下载地址：

http://www.hjtag.com/download.html

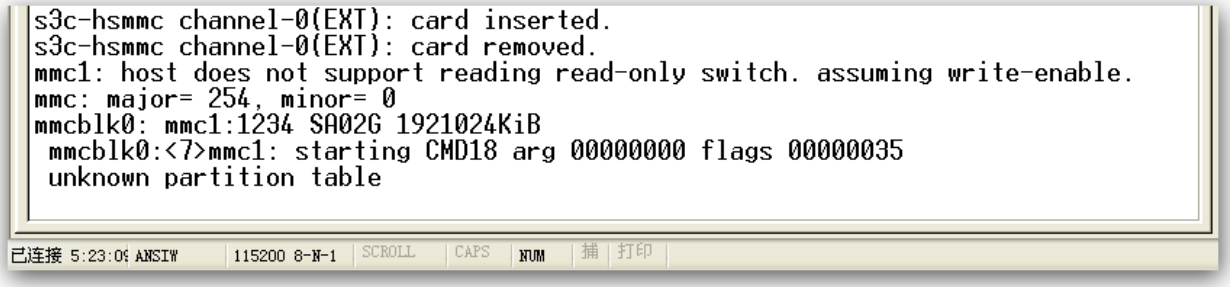
安装好后，运行 H-JTAG，开始->程序->H-JTAG->H-JTAG，如下图：



如果程序中间区域显示以上信息则表明 JTAG 接口测试通过。

2.2.3 SD 卡接口测试

S3C2416 支持两个 SD 卡接口，本开发板只做了一个 TF 卡座②和一个 SDIO 接口插座㉔，在 LINUX 系统起来后，将已经在电脑上格式化为 FAT32 的 TF 卡插入插入②中，如果串口打印类似如下信息，则测试通过：



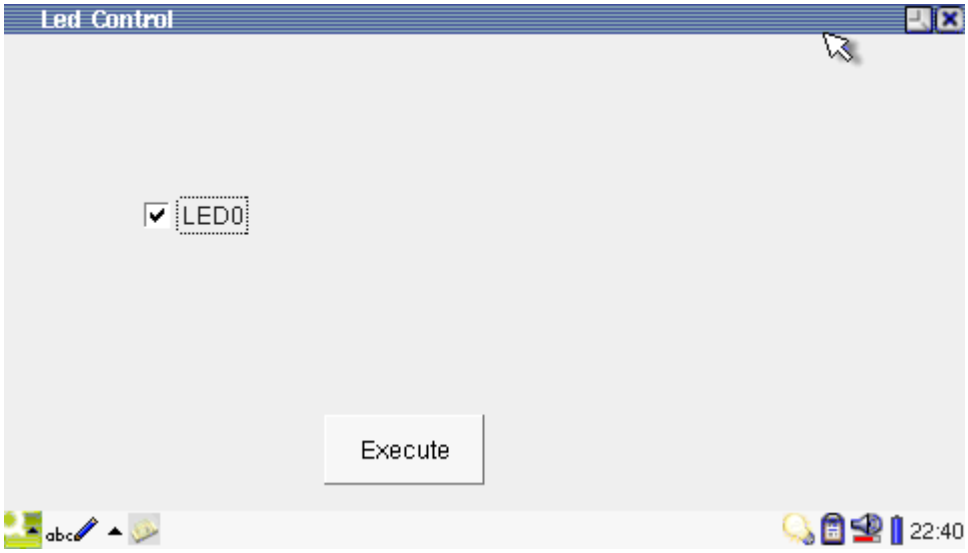
如何读写 SD 卡将在 5.2 节描述。

2.2.4 键盘测试

在 1.3 节图中的⑭，是本开发板的键盘，分别定义了上、下、左、右、回车五个键（从左到右依次排列），按上下左右可以选中某个应用程序，按回车可以执行当前应用程序。如果每个键都有动作，则测试通过。

2.2.5 可编程 LED 测试

点击 jyxtec 栏中的 led_control，运行 led_control 程序，如下图：



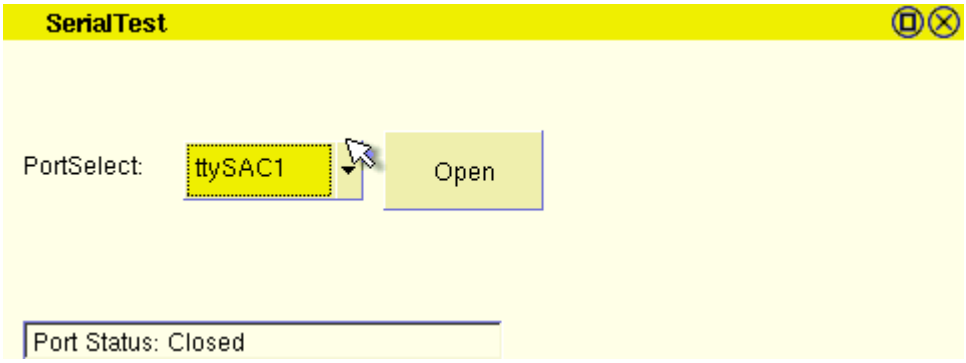
勾选 LED0，再点 Execute，开发板上的 LED 灯（1.3 节图中㉗）应该会相应的亮灭。重新选中或者取消选中，再点 Execute，如果板上的 LED 灯有相应的反应的话，则测试通过。本程序源码在软件包目录：Helper2416/source/tests/led_control。

2.2.6 串口测试

开发板配备了 4 个可编程串口，其中 COM0、COM1 可配置成 RS232 串口或者 TTL 串口，COM2、COM3 是 TTL 串口，当⑩位置的两个跳线都配置成 时，COM0 和 COM1 则为 RS232 输出，即输出到③和⑥，当⑩位置的两个跳线配置成 时，则 COM0 和 COM1 为 TTL 输出，即输出到④位置的 COM0 和 COM1，当然，也可以分别配置 COM0 和 COM1。

我们提供的软件已经包含 4 个串口的驱动。前面我们已经用 COM0 作为调试信息输出口了，如果前边的打印信息正常，说明 COM0 工作正常，以下方法不适用于测试 COM0，是用于测试 COM1~COM3 的。

点击运行 jyxtec 栏目下的 serial_test 程序，运行结果如下：



使用标准双母头交叉线连接 PC 机的串口和开发板的串口 1（1.3 节图中⑥），并打开超级终端。

SerialTest 程序的 PortSelect 选项默认是选中 COM1(即 ttySAC1)的，直接点击按钮“open “，在超级终端里会输出“hello world “字样，在超级终端里输入任意可见字符（输入回车看不出来），超级终端里应有相应的字符反回，说明串口收发正常。

注：超级终端默认是没有回显的，因此输出一串"abc"，即显示一串"abc"，而不是出现“aabbcc"。

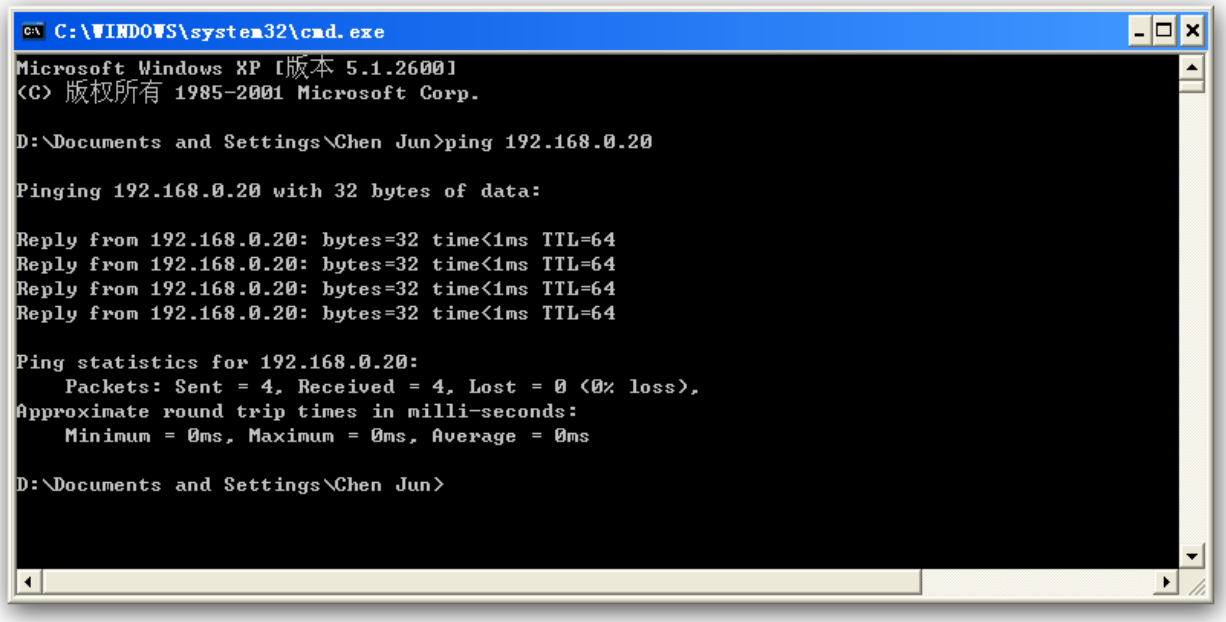
由于 COM2，COM3 没有直接输出到 RS232，不能直接与电脑相连，所以无法直接测试，当然，如果您有购买我公司提供的 TTL 转 RS232 转接板的话，一样可以按下述方法测试。

把串口线从串口 1 拨出，通过 TTL 转 RS232 转接板插到 COM2 上，点选 PortSelect 里的 ttySAC2，此时无需在点“open “按钮，在超级终端里又会输出“hello world “字样，在超级终端里输入任意字符，超级终端里应有相应的字符反回，说明串口收发正常。同样按上述方法可以测试串口 3 是否正常。

ttySAC0 对应串口 0，ttySAC1 对应串口 1，ttySAC2 对应串口 2，ttySAC3 对应串口 3。本程序源码在软件包目录：Helper2416/source/tests/serial_test。

2.2.7 网口测试

开发板的默认 IP 地址是 192.168.0.20，用普通网线连接开发板的以太网口和 PC 机网口（交换机也可以），在 PC 机上 ping 开发板的 IP 地址，如果有响应则测试通过。如下是 PC 机上的截图：

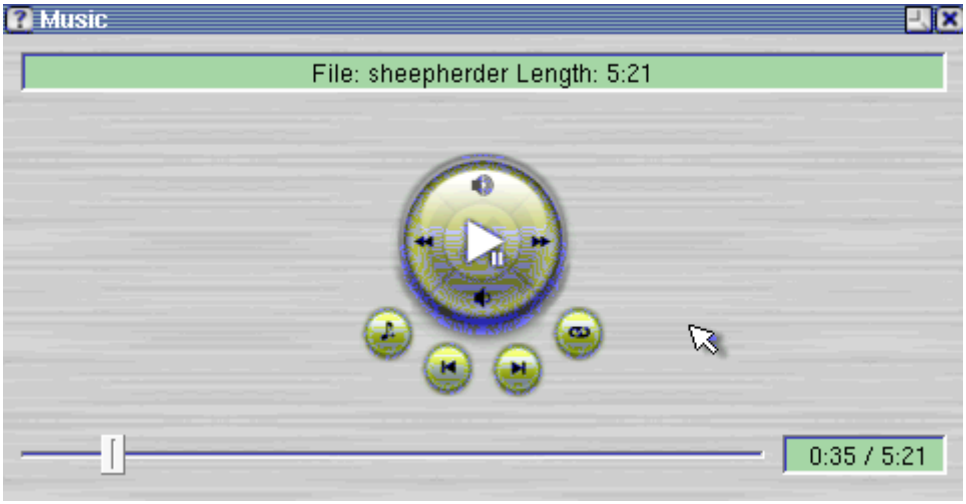


注意：PC 机的 IP 地址必须与开发板的 IP 地址在同一网段，如果不是，请修改 PC 机 IP 地址。

2.2.8 声卡测试

声音测试

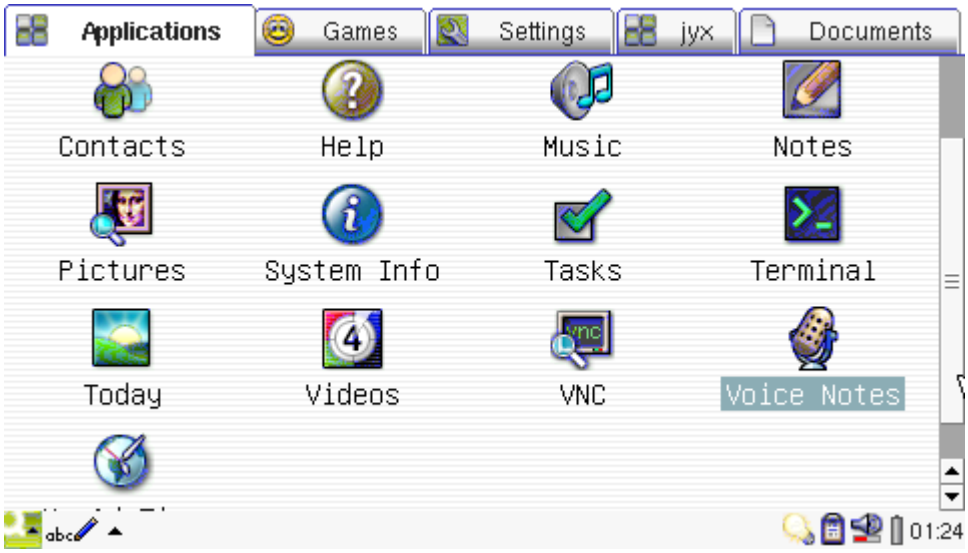
点击 Documents 栏，再点击文件 goatherder，则会自动打开音乐播放程序，播放音乐。如下图：



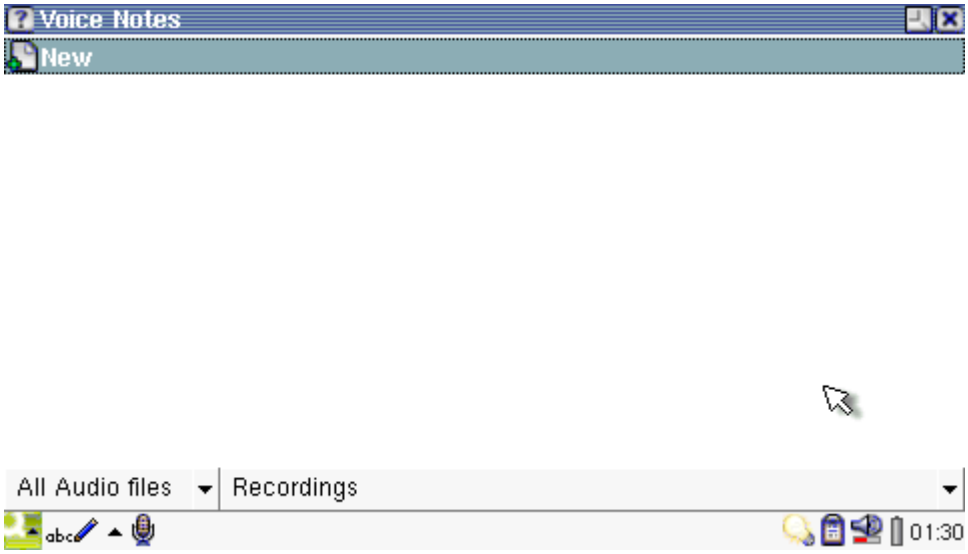
插上耳机，如果您听到的是云朵的《孤独的牧羊人》音乐，那么恭喜你，说明声音输出工作正常，享受一下这优美的歌声吧。

麦克风测试

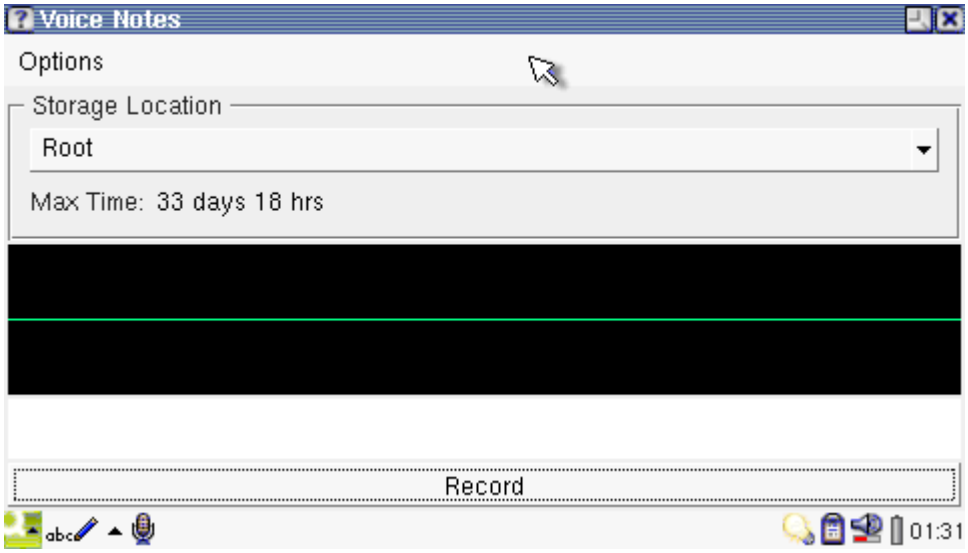
点击 Applications 栏目里的 Voice Notes，如下图：



运行结果如下：

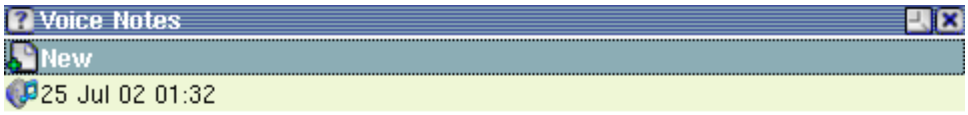


点击 New，启动录音机：



再点击 Record 按钮，开始录音，对着开发板上的麦克风（1.3 节图中⑧）说：“只谈买卖，不谈感情。 ”

再按 Stop 按钮，则会退出录音程序，同时生成一个以时间为文件名的声音文件，再点击该声音文件，就会启动声音播放程序，如果声音播放正常，说明麦克风工作正常。如下图：



2.2.9 USB Host 测试

在 USB Host 接口（1.3 节图中①）插入 U 盘或者键盘、鼠标，串口 0 会有相应的打印信息，表明 USB Host 工作正常。

2.2.10 USB Device 测试

使用普通一端小头（MicroUSB）一端标准头的线连接开发板的 MicroUSB 接口（1.3 节图中⑳），可参考 5.4 节的方法进行测试。

2.2.11 E2PROM 测试

可用 eeprog 做测试，具体方法自行了解。

2.2.12 CAN 总线测试

软件已实现，但由于需要两台 CAN 总线设备才能测试，具体方法就不在此介绍了，当然，两块开发板对传也可以，有兴趣可以到网上了解 canutils 的使用方法，我们提供的根文件系统已经包含 canutils。

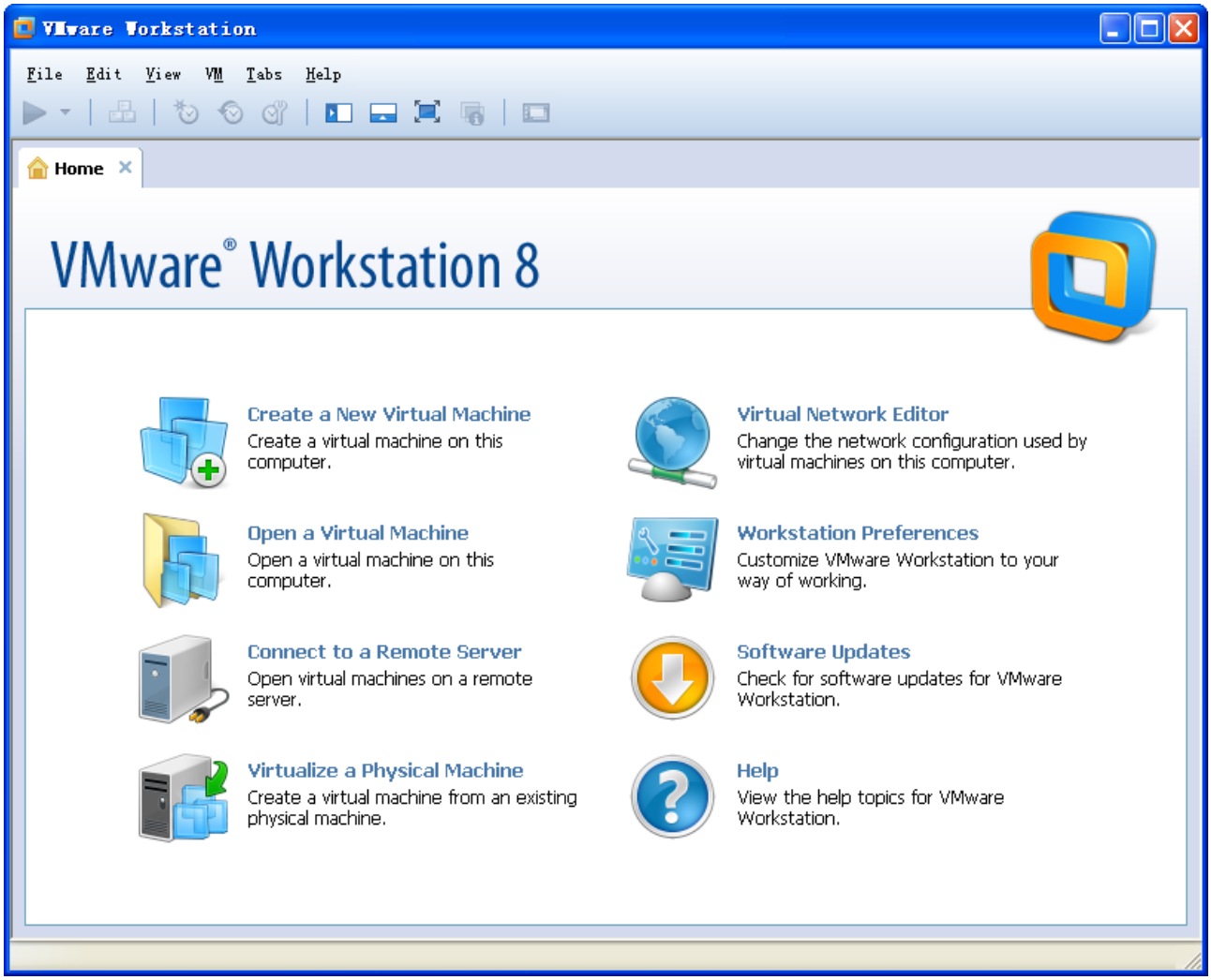
3 开发板平台构建

3.1 使用 vmware 虚拟机镜像构建虚拟机开发环境

工具：vmware 8.0(据说 9.0 的性能更好)，光盘目录：Helper2416/tools 里有提供安装文件和序列号。请自行安装。

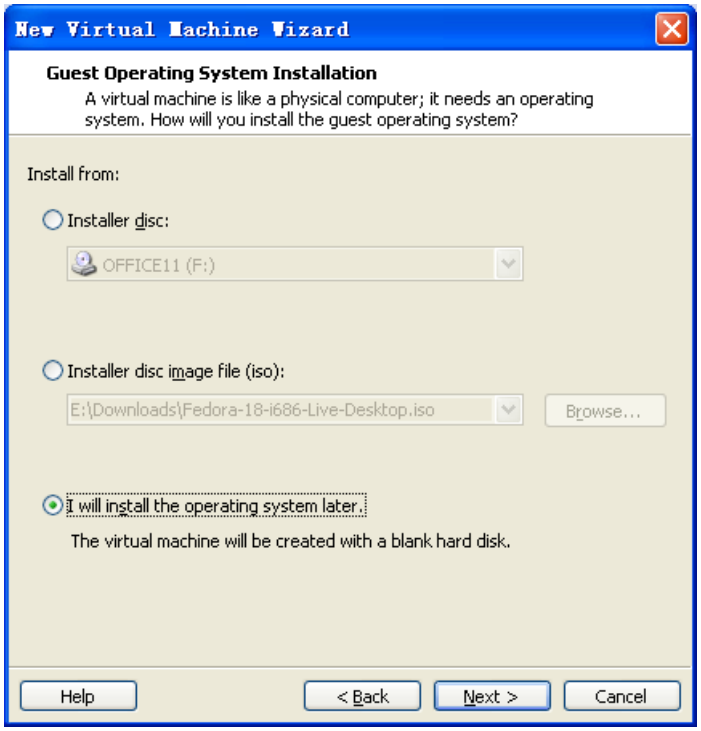
特别注意：新一点的 CPU 都支持 visualization engine，需要在 BIOS 里手工设置使能，设置该项之后，虚拟机的运算速度会如同真机。

安装好 vmware，并打开 vmware workstation，界面如下图：

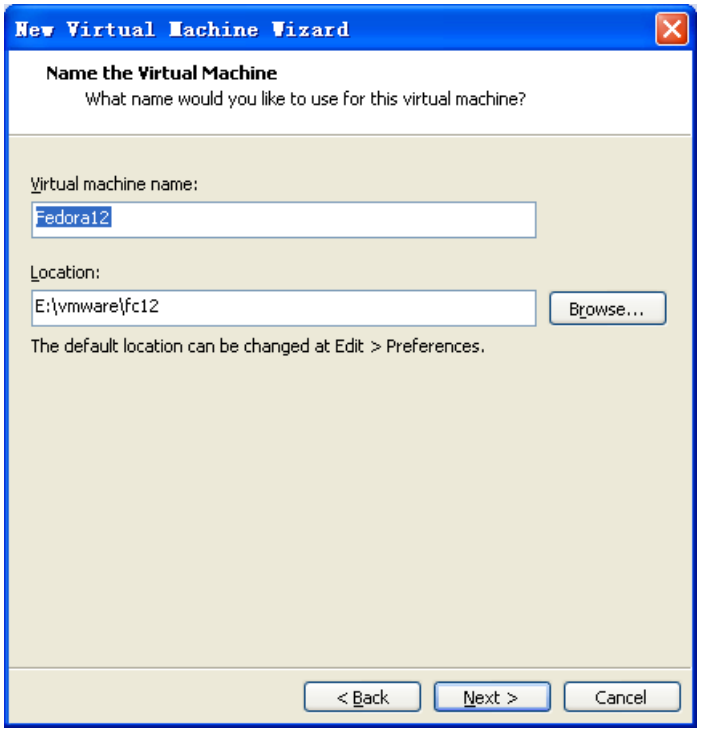


点击 File->New Virtual Machine：

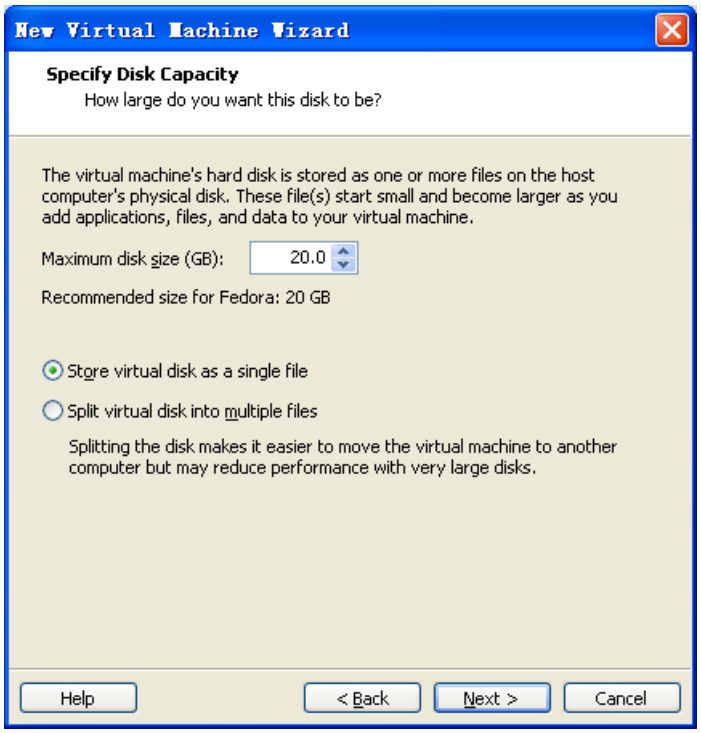
选择 Typical 点击 Next：



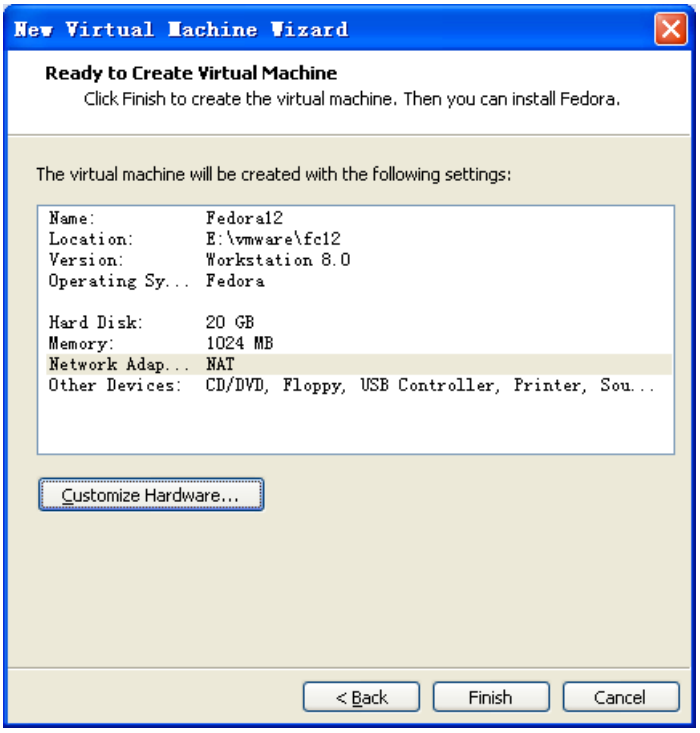
上边选 Linux，下边选 Fedora，再点击 Next，并输入操作系统名称 Fedora12 和系统存贮位置，如:E:\vmware\fc12：



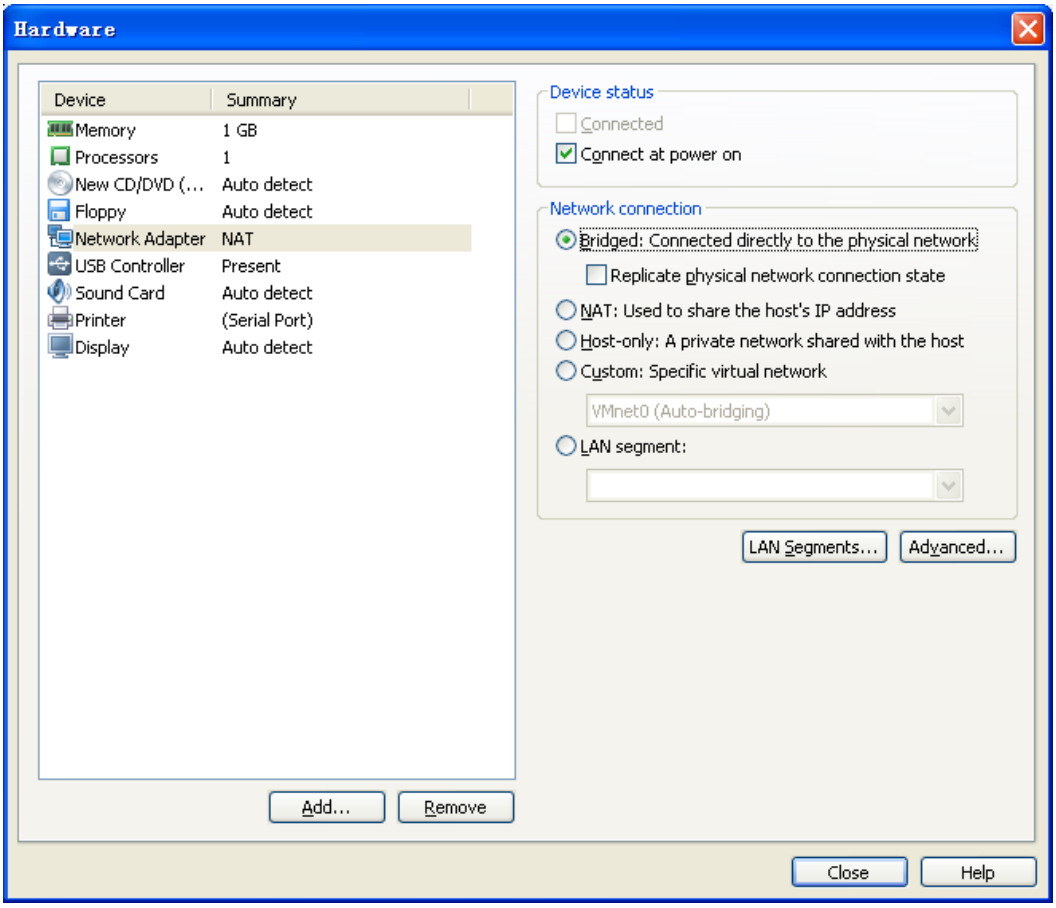
再选择：Store virtual disk as a single file（需要装到 NTFS 分区上， 否则单个文件不能大于 4G），此处大小不用考虑，因为后边我们不用这个配置：



然后点击 Next 下一步：

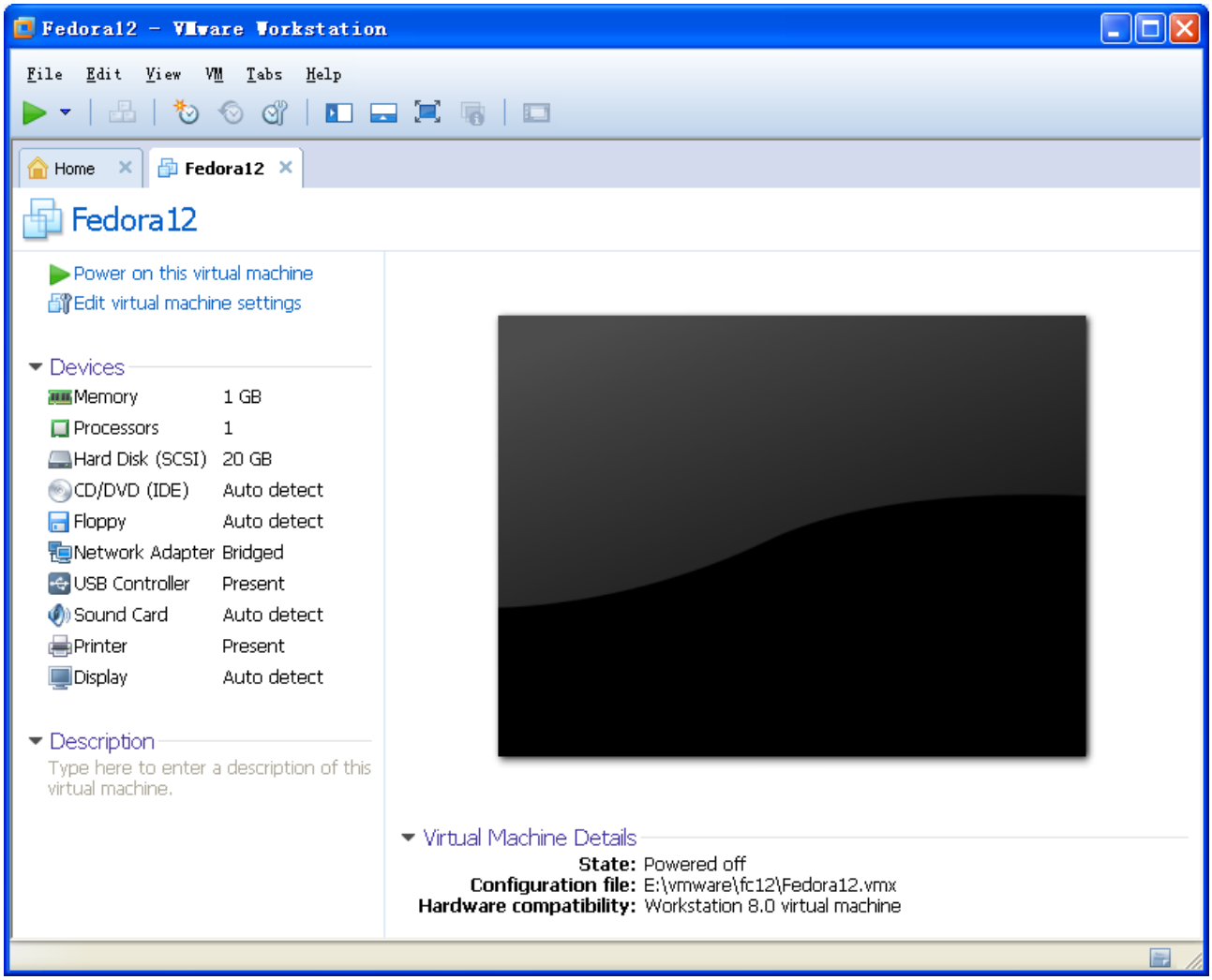


再点击 Customize Hardware，将 Network Adapter 设置为 Bridged 模式：



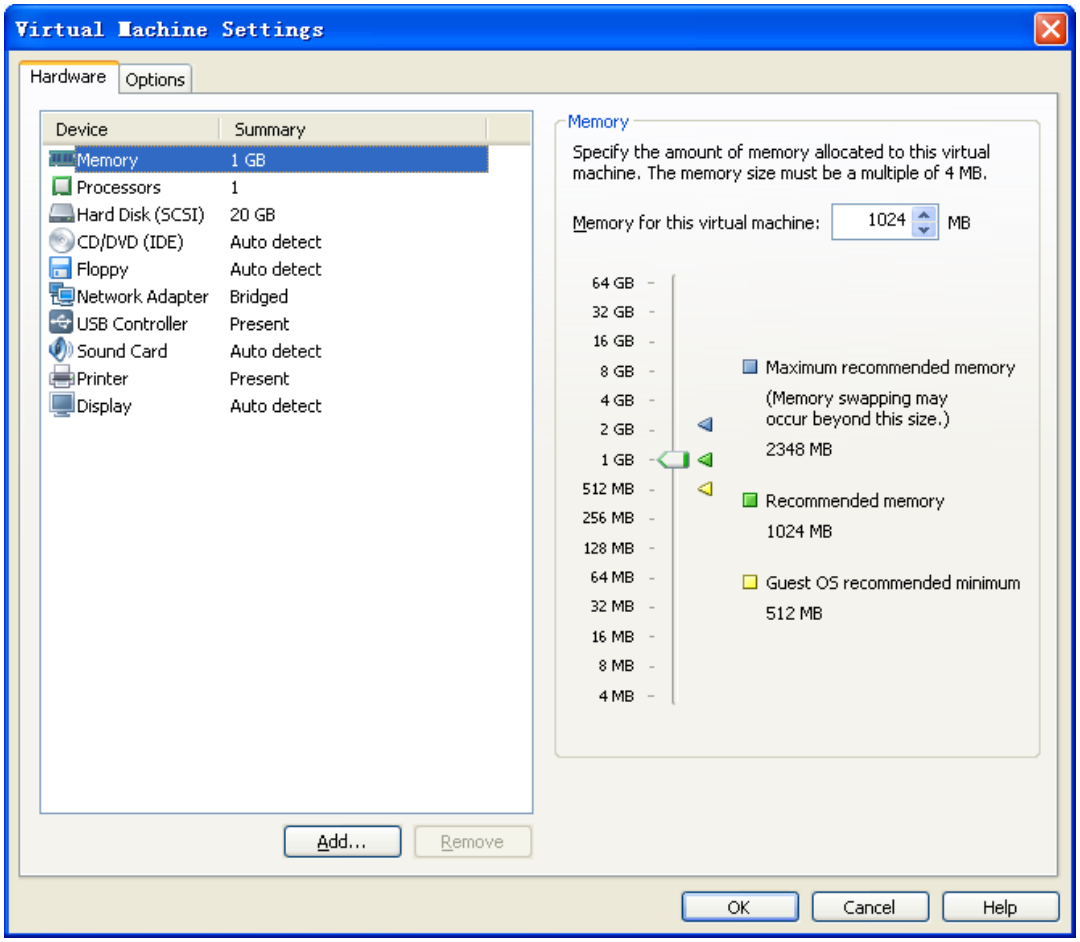
说明：因为我们的板子可能会访问虚拟机，所以此处最好是用桥式配置。

然后点击 Finish 完成，回到虚拟机软件 vmware 主界面：

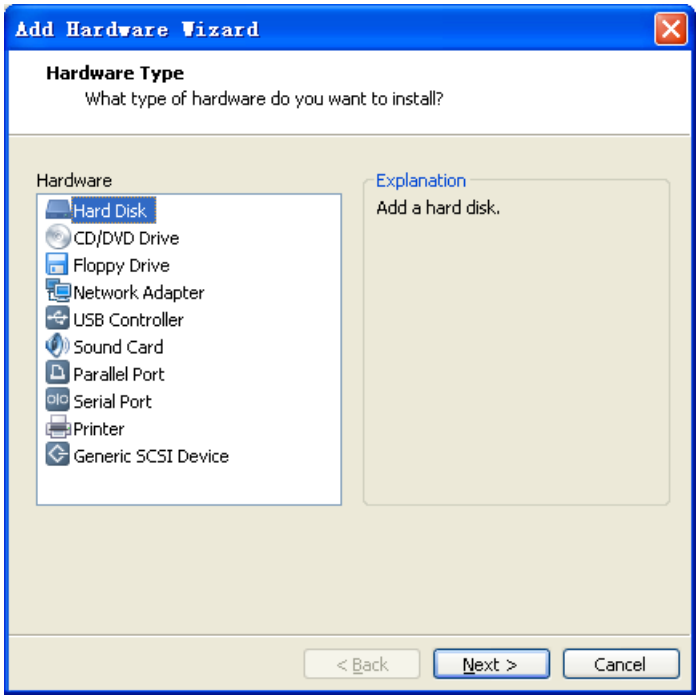


光盘目录里的：Helper2416/tools/fc12.7z，是基于 vmware 8.0 的虚拟机硬盘镜像压缩文件。此时请解压缩到本地硬盘安装虚拟机系统的目录，此处是：E:\vmware\fc12\fc12.vmdk。

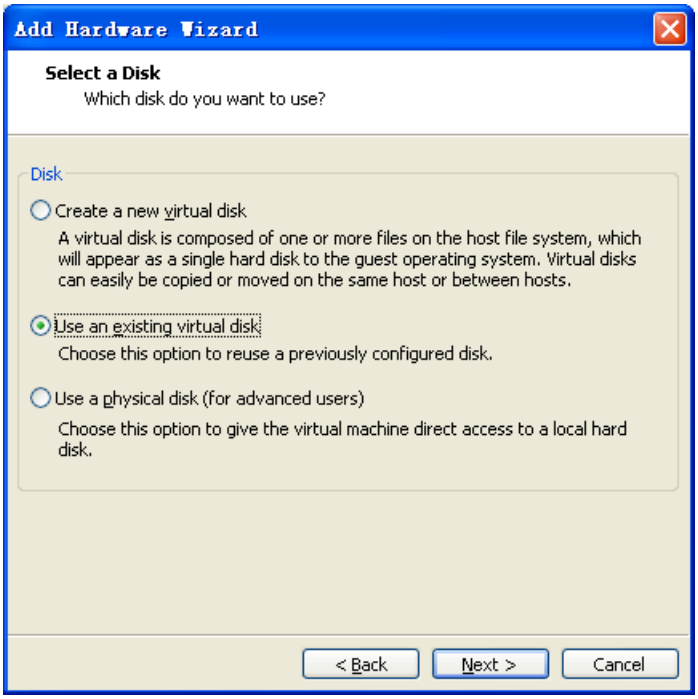
然后在虚拟机 vmware 的 vm 菜单，点击 Settings：



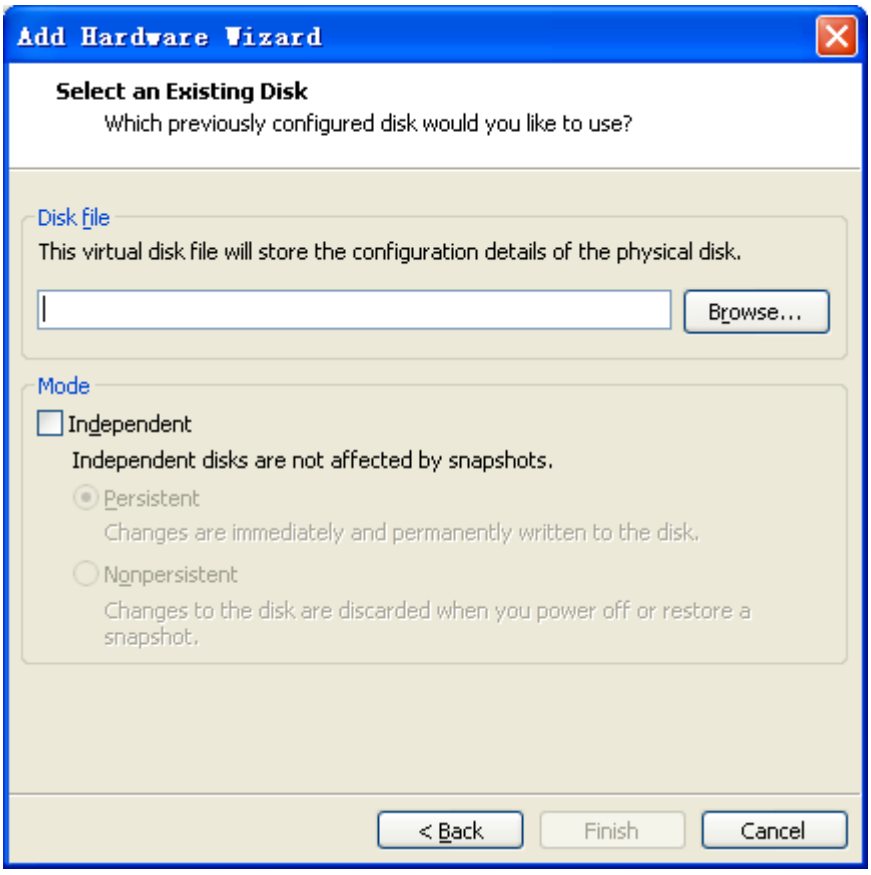
选中 Hard Disk，点击下边的 Remove，移除原来的硬盘配置，再点击 Add...，选中 Hard Disk，



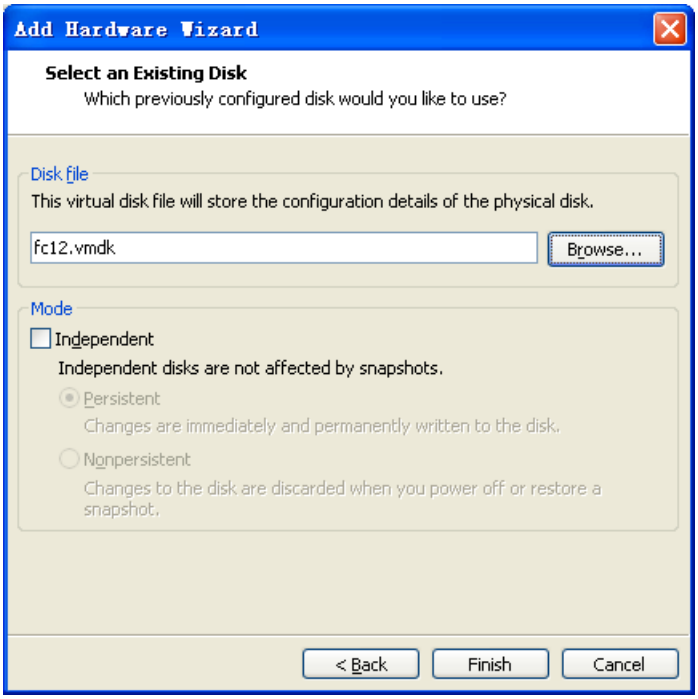
点击 Next 下一步，选择 Use an existing virtual disk：



点击 Next 下一步，再点 Browse：

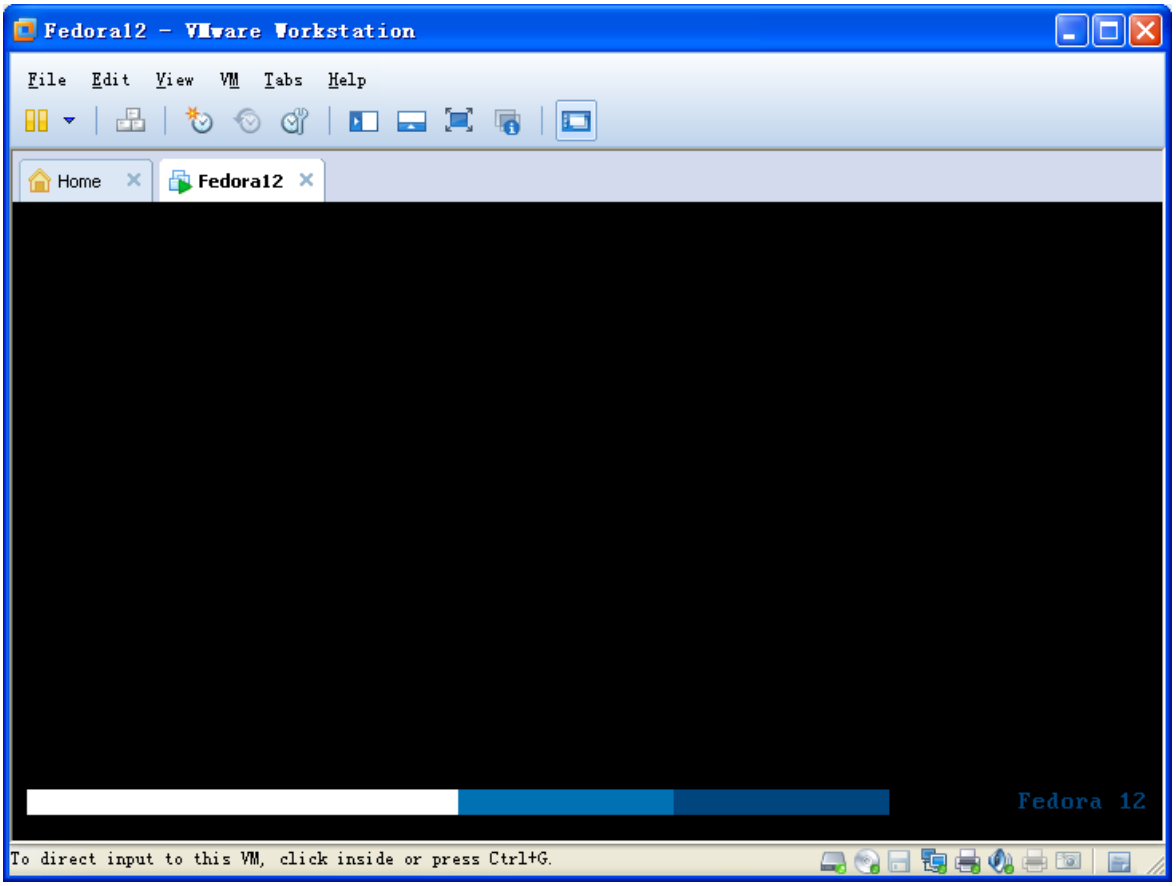


选择我们刚刚解压缩出来的 fc12.vmdk：

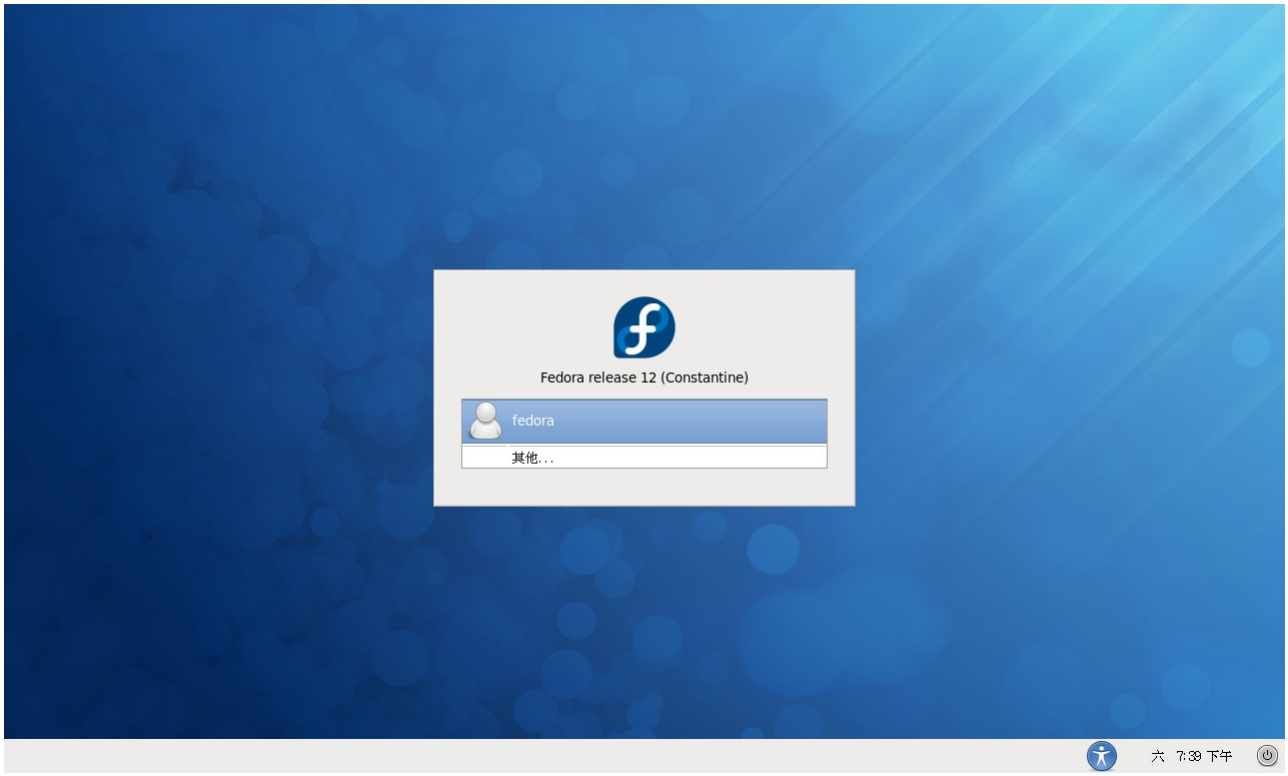


点击 Finish 完成，再点 OK。

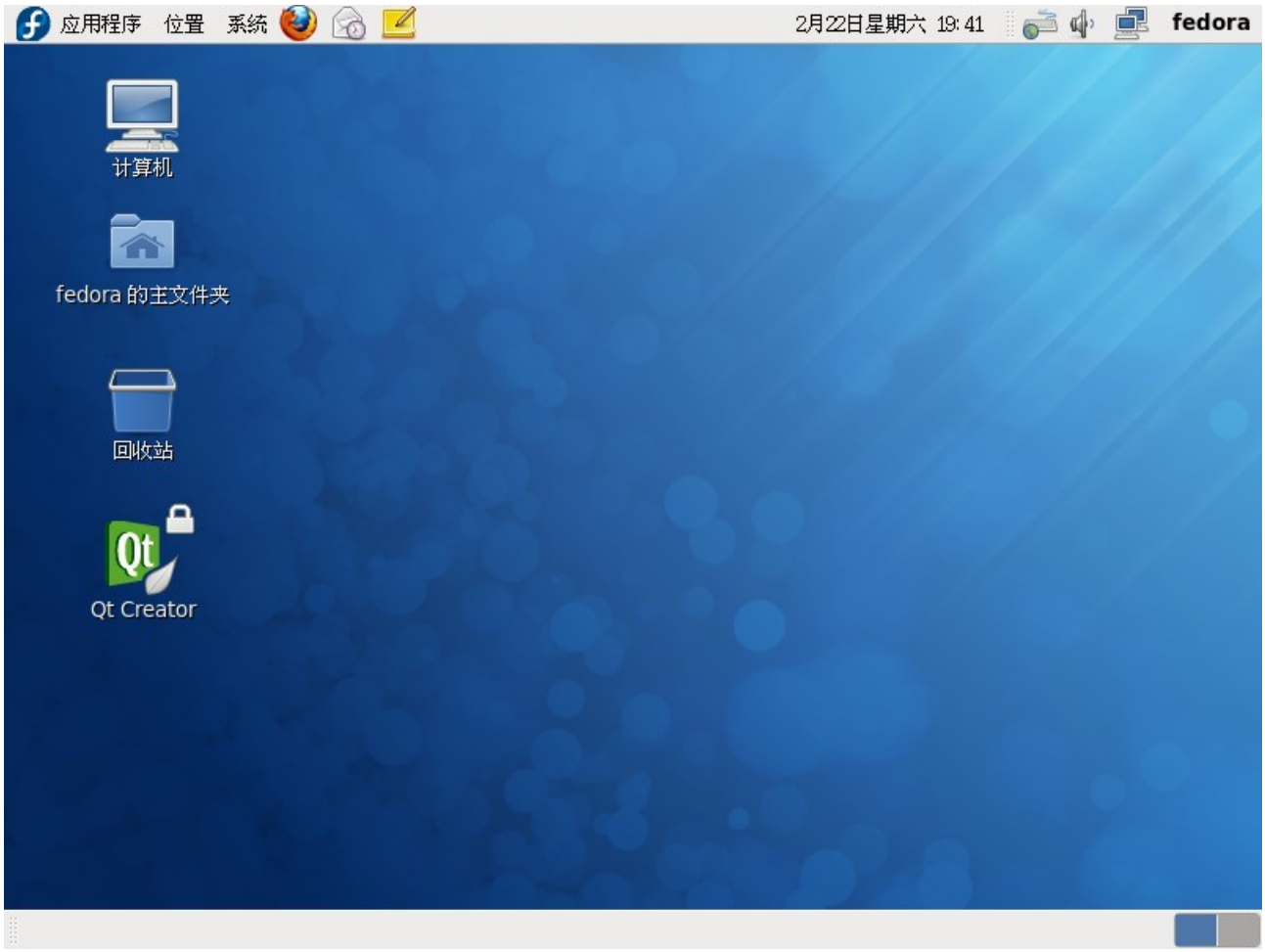
点击界面工具栏左边的绿色三角形按钮，power this virtual machine，Fedora 12 系统开始启动：



稍候进入如下界面：



点选 fedora 用户，再输入：fedora 作为密码，进入该 LINUX 系统。



到此为止，基于 Fedora 12 的虚拟机开发环境安装完成，该虚拟机里已经安装好了我们提供的编译器，

qt4.7.2 的嵌入式系统库等等，您已经可以直接在这个虚拟机里编译基于 Helper2416 的嵌入式 LINUX 软件了。

Fedora 12 是一个非常稳定的 Linux 发行版，我们一直在该发布版下开发 LINUX 应用，该发行版对于一般的开发也是足够的了。而且体积也相对较小，比较便于我们发布，当然，如果你有兴趣，可以自行安装别的发行版，或者更新的 Fedora，我们将在附录 8.1 里介绍如何安装一个完整的虚拟机操作系统。

3.2 烧写 LINUX 系统到板载 NAND Flash

烧写 LINUX 系统的方法很多，下面只介绍从 TF 卡烧写到 NAND Flash 的方法。

3.2.1 烧写系统镜像到 TF 卡

请使用 2G 的足量 TF 卡或者更大容量的足量卡，低于 2G 容量只能烧写 u-boot，烧写内核等文件需要手动完成，非足量卡不能用于启动，本公司销售的 TF 卡是保证支持启动的。

用到的映像文件：

SD BootLoader: u-boot-movi.bin；（TF 卡启动时所用）

Nandflash Bootloader：u-boot.bin；（nandflash 启动时所用）

Linux Kernel：zImage；（内核映像文件）

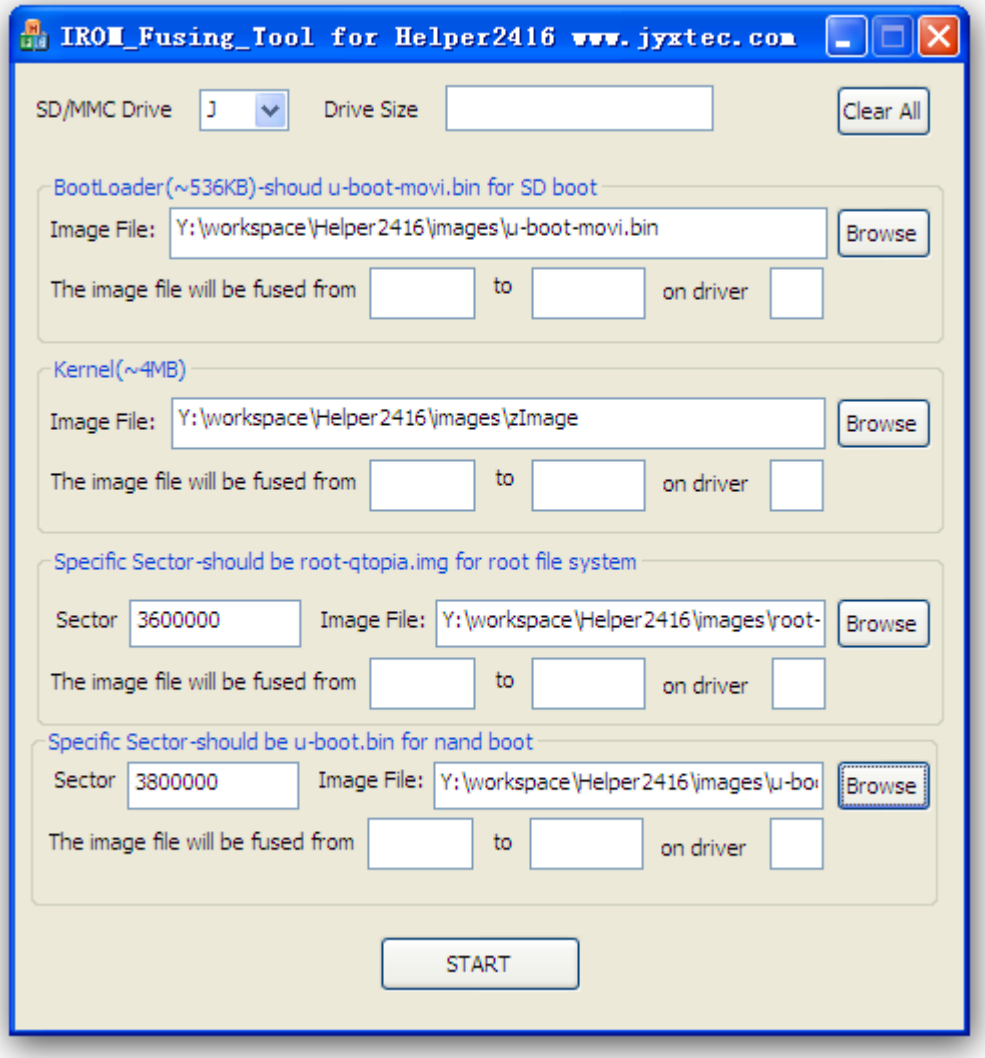
Root file system： root-qtopia.img；（文件系统映像文件）

它们全部都被放置在软件包的 Helper2416/image 目录，下面介绍将这些文件烧写到 TF 卡，以便后边通过 TF 卡烧写到开发板的内部 NAND Flash 中。

打开软件包的 tools 目录中的 IROM_Fusing_Tool.exe，将 SD 卡放入读卡器，然后插入电脑。在电脑上找到 SD 卡盘符后，再切换到 IROM_Fusing_Tool.exe 界面：

1) 在第一行 SD/MMC Drive 右侧选择 相应的 SD 卡的盘符
2) 点击 Bootloader 右侧的 Browse， 找到镜像中的 u-boot-movi.bin，并选中。
3) 点击 Kernel 右侧的 Browse，找到镜像中的 zImage(4.3 寸液晶屏选 zImage.43，5.6 寸液晶屏选 zImage.56)，并选中。
4) 点击 Specific Sector - root-qtopia.img 右侧的 Browse，找到镜像中的 root-qtopia.img，并选中
5) 将 Sector 设置为 3600000，如下图：
6) 点击 Specific Sector - u-boot.bin 右侧的 Browse，找到镜像中的 u-boot.bin（也可能是 u-boot.534.64m.bin 等），并选中
7) 将 Sector 设置为 3800000，如下图：

说明：u-boot.534.64m.bin：表示 CPU 的频率是 534MHz，内存是 64MB，相应的 u-boot.600.128m.bin，表示 CPU 的频率是 600MHz，内存是 128MB。



点击 START，稍等片刻（20 秒到 1 分钟，不同的卡时间不同），如果烧写成功，则会弹出如下窗口：

如果写入不成功，请检查一下 SD 卡是否写保护了，或者重新拔插一下再试，如果是 win7 系统可试试右键点击 IROM_Fusing_Tool.exe，选择“以管理员身份运行”。

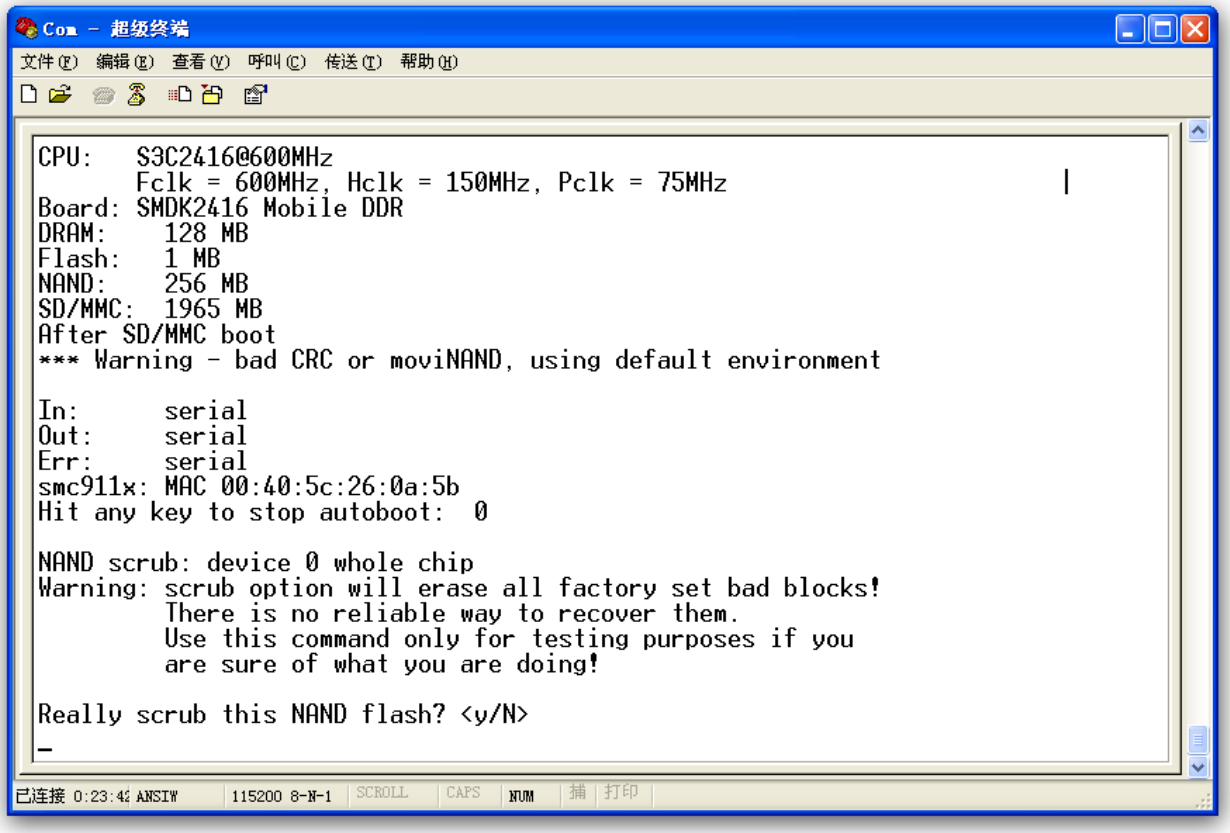
安全删除读卡器设备，取出 SD 卡，放到开发板的 SD 卡槽中（1.3 节图中②）。

说明：本公司提供的 SD 卡烧写工具可以单独烧写任何文件到 SD 卡指定位置，如单独烧写启动文件 u-boot-movi.bin，就只选 BootLoader 一项即可。

另外，烧写是直接写到 SD 卡指定扇区的，烧写完成后，SD 卡里是看不到文件的。

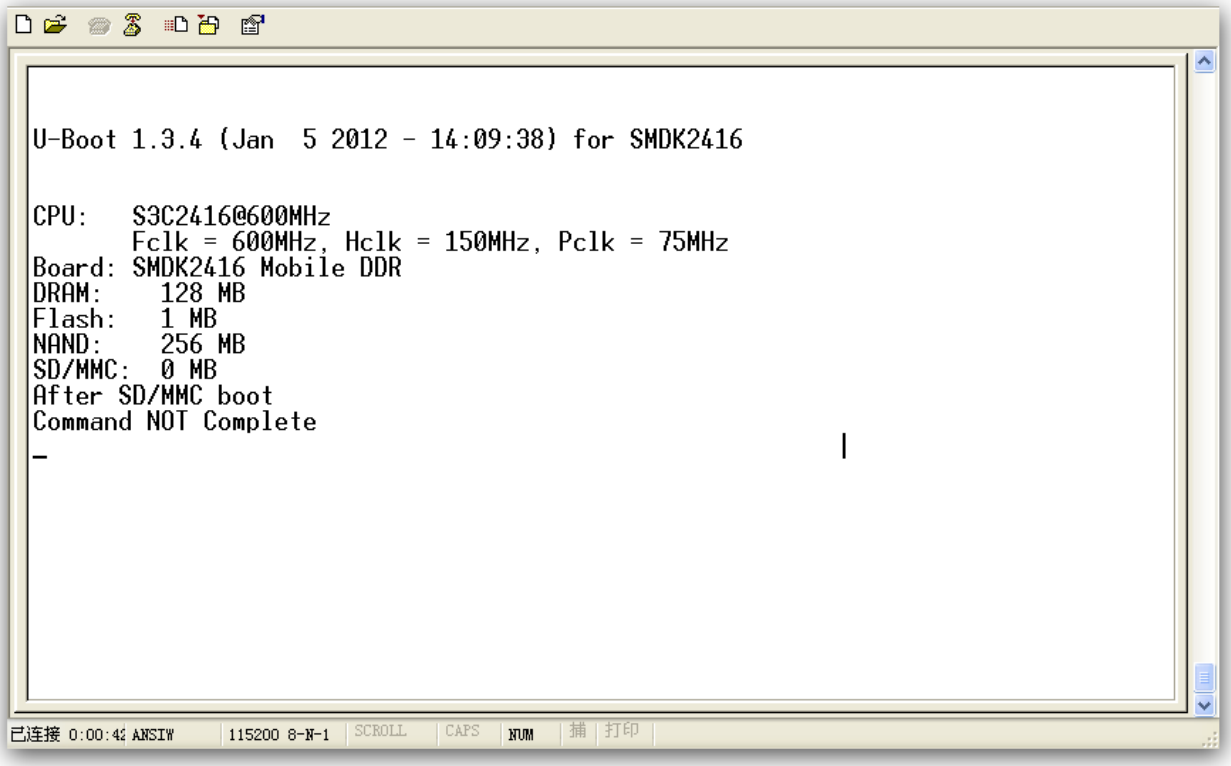
3.2.2 从 SD 烧写 LINUX 系统到 NAND Flash

打开超级终端，打开 2.1.4 节保存的文件，在菜单中的点击 文件->打开->选择 jyx.ht，把烧好数据的 SD 卡插入板子的 SD1 卡座，拨动启动模式开关全部到 ON 位置，即 SD 卡启动端（参见 2.1.1 节板子启动模式设置），接通开发板电源，出现如图信息：



按 y 并回车（此处是完全擦除 NAND，不要多按，多按跳过，按错了只能重新启动，否则会跳过而失败），会自动把 SD 卡中的 u-boot.bin，zImage，root-qtopia.img 写到 nandflash 相应位置，大约需要两三分钟的时间。

注：如果出现如下情况，则说明 SD 卡质量较差，不能承受较高的时钟速率，系统虽然能从 SD 卡引导，但是后边的进一步读取则不能满足要求，建议换卡。



烧写完后，关闭电源，拔动启动模式开关到 NAND 启动模式，打开电源，板子会从 nandflash 启动，此后，可以重复 2.2 节的测试。

注：板子启动时，u-boot 默认会有三秒的时间接受按任意键（是对应串口输入的，非板子上的键）进入 u-boot 命令行模式，有兴趣的朋友可以尝试一下，u-boot 有很多命令，可以完成如读 SD 卡，读 NAND 的命令，命令行下输入 help 可以得到 u-boot 支持的命令，help 后面跟 u-boot 命令会打印此命令更详细的信息，如 help nand。

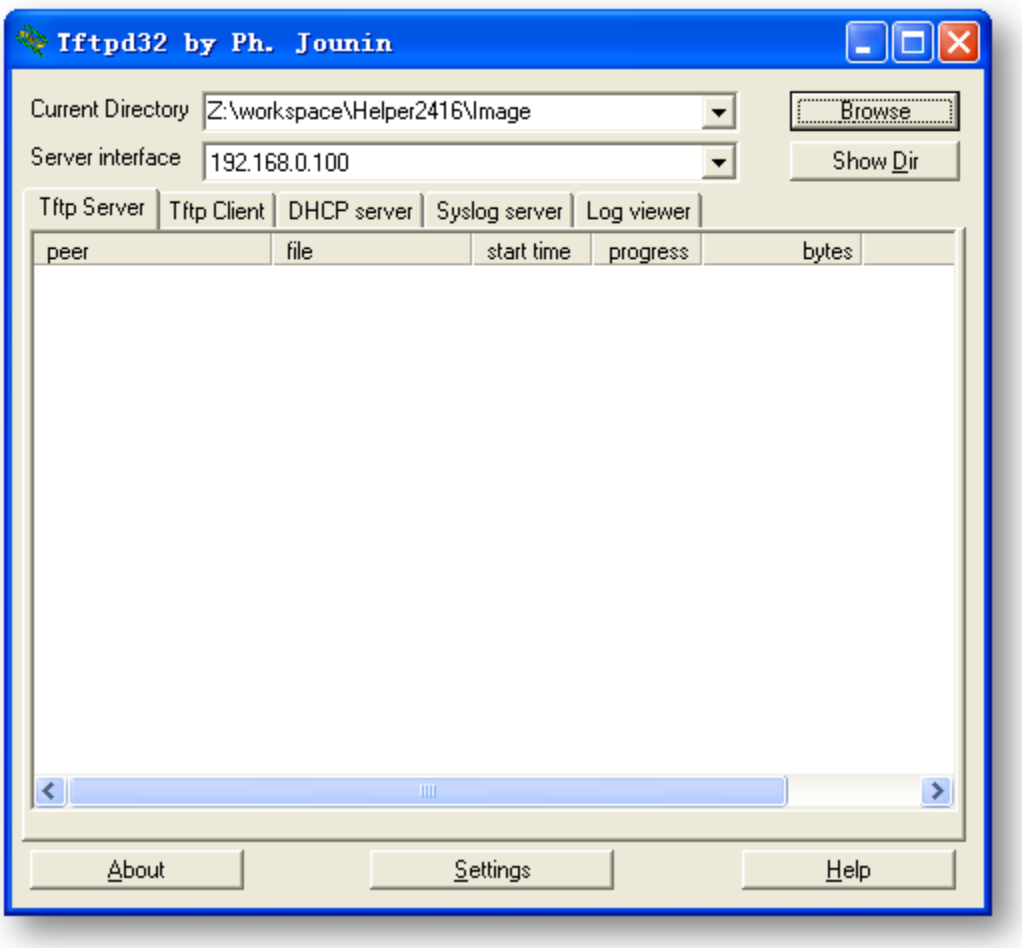
3.3 用 tftp 搭建板子系统

TFTP 是一个基于 UDP 协议的文件传输工具，我们可以使用 TFTP 工具将 u-boot.bin，zImage 等各种文件传到开发板的内存，然后烧写到 NAND。如果您的 SD 卡小于 2G，则可以只烧 u-boot-movi.bin 到 SD 卡，然后用下列方法再烧到 NAND 中，下面介绍 tftpd32 的使用方法。

3.3.1 启动 tftp 服务

3.3.1.1 针对 windows

如果是使用 windows 做 tftp 服务器，则运行软件包中，Helper2416/tools/tftpd32.exe，防火墙放行，如下图(win7 用户可自行搜索 tftpd32 win7 下载，比如华军软件园，并右键选择以管理员身份运行，防火墙放行)：



Current Directory 是服务器的根目录（u-boot.bin 等文件放在这里），点击 Browse 可以设置；Server interface 是服务器 ip，会自动获取的。

3.3.1.2 针对 LINUX

如果是使用 LINUX 作为 tftp 服务器，则需要安装 tftp-server 包，以 fedora 12 为例：

```
$ sudo yum install tftp-server -y
```

创建文件：/etc/xinetd.d/tftp，内容如下：

```
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.   The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
```

```
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

其中 server_args 是 tftp 服务目录，此处设为根目录下的/tftpboot，拷贝文件到该目录，在 u-boot 里可以访问得到。我们需要手动创建该目录：

```
$ sudo mkdir /tftpboot
$ sudo chown -R fedora /tftpboot
```

后边一行是将/tftpboot 目录设置为 jyx 用户所有，将来就可以直接拷文件进去而不用输入 sudo 了。

然后运行重启 xinetd 服务，运行 tftp 服务器：

```
$ sudo /etc/init.d/xinetd restart
```

这样，LINUX 的 tftp 服务器就正常工作了，安装设置比 Windows 里麻烦一点，但是编译都是在 LINUX 下进行，在 LINUX 下拷贝文件更方便。

3.3.2 设置 u-boot 环境变量

板子接电源启动 u-boot，三秒内按任意键进入命令行，可设置板子 ip，服务器 ip，网关等，其中，serverip 要设置成运行 tftp 服务器的电脑 IP。

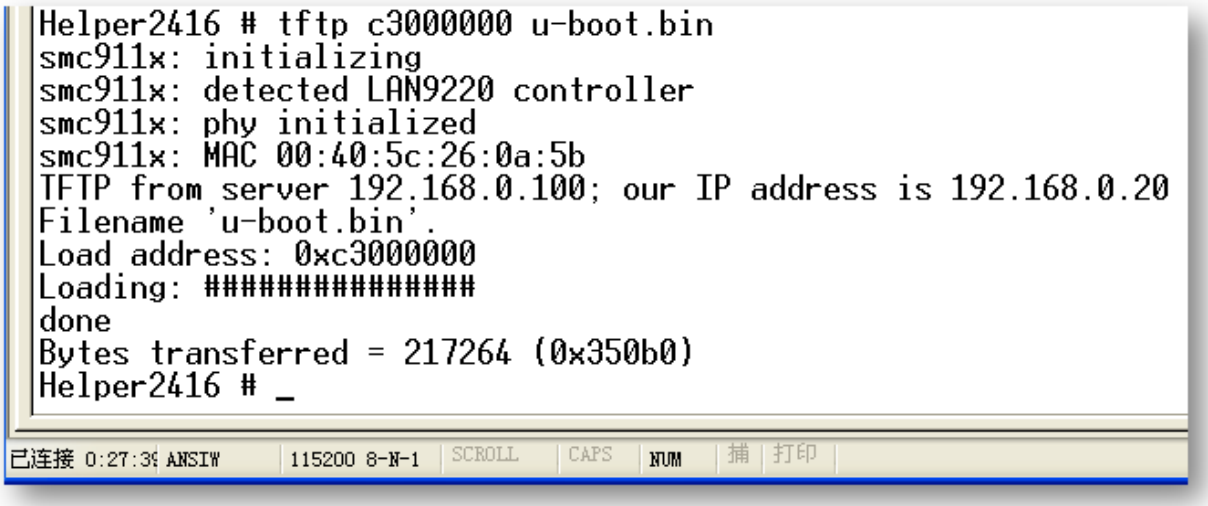
```
# set ipaddr 192.168.0.20
# set serverip 192.168.0.100
# set gateway 192.168.0.1
# save
```

ipaddr 代表开发板的 IP，serverip 代表运行 tftp 服务器的 IP，gateway 代表网关，局域网内可以不设网关。

3.3.3 烧写 u-boot

加载 u-boot.bin

```
# tftp c0000000 u-boot.bin
```



先对要写的位置擦除，0~40000 区域，16 进制，即 NAND 的前 256KB

```
#nand erase 0 40000
```

把 u-boot.bin 烧写到 nandflash 中

```
#nand write c0000000 0 40000
```

3.3.4 烧写 kernel

zImage 文件拷贝到 tftp 指定的数据存放目录，再下载到开发板，并烧写

```
#tftp c0000000 zImage
# nand erase 40000 3c0000
# nand write.i   c0000000   40000   3c0000
```

指定从 nand flash 中加载 kernel

```
#set bootcmd 'nand read.i c0008000 0x40000 0x400000;bootm c0008000'
#save
```

nand read 后边的“.i”表示从自动跳过坏块。

3.3.5 烧写根文件系统

root-qtopia.img 文件拷贝到 tftp 指定的数据存放目录

```
# tftp c0000000 root-qtopia.img
# nand erase   400000
# nand write.yaffs c0000000 400000      $filesize
```

指定从 nandflash 中挂载根文件系统

```
# set bootargs      root=/dev/mtdblock2 console=ttySAC0，115200
#save
```

再运行 boot 命令即可启动开发板，当然，如果您是用 SD 卡启动并烧写系统的，则需要选择从

NAND 启动，再重启

```
# boot
```

4 源码编译

如果您购买了软件包光盘，则直接拷贝 Helper2416 目录到 LINUX 系统，如果您是直接下载的软件包，则将软件包拷到 LINUX 系统任意位置，解压缩

```
$ tar -xvf Helper2416.tar
```

4.1 编译 u-boot

先在用户目录下创建一个工作目录

```
$ mkdir ~/workspace
```

把 u-boot 源码包解压到~/workspace 目录下

```
$ tar -xvf   ./Helper2416/source/u-boot-1.3.4.tar.xz -C /home/fedora/workspace/
```

默认的代码编译生成的是在 nandflash 中启动的二进制文件 u-boot.bin，可将该文件烧写到 nand flash，实现 nand 启动。

```
$ make smdk2416_config
$ make
```

得到 u-boot.bin，再拷贝到/tftpboot 目录，可按 3.3.3 节的方法烧写进 nand flash。

```
$ cp u-boot.bin /tftpboot
```

如果要生成引导 SD 卡启动的 u-boot-movi.bin 得进行修改：

用编辑器(vi 或者 gedit)打开 u-boot 源码目录下的 include/configs/smdk2416.h

大约在 254 行处，如下代码：

```
/* Boot configuration (define only one of next) */
//#define CONFIG_BOOT_NOR
#define CONFIG_BOOT_NAND
//#define CONFIG_BOOT_MOVINAND
//#define CONFIG_BOOT_ONENAND
```

将

```
#define CONFIG_BOOT_NAND
//#define CONFIG_BOOT_MOVINAND
```

改成

```
//#define CONFIG_BOOT_NAND
#define CONFIG_BOOT_MOVINAND
```

在 u-boot 源码所在文件夹下执行命令

```
$make distclean
$ make smdk2416_config
$ make;./mkmovi
```

```
$ cp u-boot-movi.bin /tftpboot/
```

得到 u-boot-movi.bin，可按 3.2.1 节的方法烧写到 SD 卡，用户从 SD 卡启动。

另外在约 443 行处定义了 SD 卡自动烧录命令的宏

```
#define CONFIG_BOOTCOMMAND  "sleep 1;nand scrub ;sleep 1;nand erase;sleep 1;movi read 3800000# 40000 c0000000 ;sleep 1;nand write c0000000 0 40000;sleep 1;movi read kernel c0000000;sleep 1;nand write c0000000 40000 300000;sleep 1;movi read 3600000# 4161B40 c0000000;sleep 1;nand write.yaffs c0000000 400000 4161B40"
```

其中最后的 4161B40(十六进制)是根文件系统镜像的大小,如果自己定制根文件系统可以修改此值。

说明：如果您用的是 128M 版本的内存，编译 uboot 之前请先打补丁：helper2416/ram64mto128m.patch，先在命令终端进入 uboot 目录，再从光盘拷贝补丁文件，执行如下命令：

$ patch -p1 < ram64mto128m.patch

4.2 编译内核

4.2.1 编译内核版本 3.2

先解压缩软件包里的内核源码：

```
$ tar -xJvf   Helper2416/source/HELPER2416-KERNEL3.2-V110.tar -C /home/fedora/workspace/
$ cd   ~/workspace/s3c-linux.jyx
```

注意：文件是用 xz 格式压缩的，解压命令里有一个大写的’J’，否则解不开。

然后看光盘的 Helper2416/source 目录和上我公司网站是否有内核更新补丁，网站查看地址：

http://www.jyxtec.com/index.php/downloads

如果有，请及时打上，补丁按数字顺序，比如：00-kernel-xxx-xxx.patch 先打，然后是 01-kernel-xxx-xxx.patch。在终端模式下，进入内核代码目录，如下：

```
$ patch -p1 < 00-xxx.patch
$ patch -p1 < 01-xxx.patch
```

然后编译内核：

```
$ make clean
$ make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

编译 kernel 后会在 s3c-linux.jyx/arch/arm/boot 目录下生成 zImage，可用于烧写到 NAND，烧写方法前面 3.4.4 节已经讲过。

4.2.2 特别说明

编译内核时，请先确认你所使用的液晶屏的大小，然后按照 7.4 节所描述的方法选择相应的液晶屏配置，否则编译的内核运行后可能不能正确显示。

4.3 编译模块

在我们提供的内核配置文件里已经选中了部分模块，如 usb gadget file backed storage，用以下方法就可以编译：

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- modules
```

生成的模块里有一个 drivers/usb/gadget/g_file_storage.ko，是 U 盘驱动，在 5.4 节会用到。

4.4 修改开机画面

我们提供的内核里包含的是我们制作的开机画面，下面介绍如何定制自己的开机画面。

准备好开机画面图片，尺寸最好是屏幕大小，保存为 bmp 格式，比如：logo.bmp，用 WINDOWS 里的画图软件可以将多种图片（如 JPG）保存为该格式。

然后在终端运行我们提供的 logo 制作脚本，在我们提供的内核目录：s3c-linux.jyx/logo_maker.sh

```
$ ./logo_maker.sh logo.bmp logo_out.ppm
bmptoppm: Windows BMP， 480x272x24
bmptoppm: WRITING PPM IMAGE
ppmquant: making histogram...
ppmquant: 188 colors found
ppmquant: choosing 224 colors...
ppmquant: mapping image to new colors...
```

即生成内核可识别的 logo 图片：logo_out.ppm，拷贝到内核目录并覆盖原来的文件：./kernel2416/drivers/video/logo/logo_linux_clut224.ppm

```
$ cp logo_out.ppm drivers/video/logo/logo_linux_clut224.ppm
```

然后在重新编译内核：

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

即可得到包含定制开机画面的内核 arch/arm/boot/zImage，重新烧写到 nandflash 内核区域，重启开发板即可看到效果。

注：logo_maker.sh 的输出文件，一定不能是文件名 logo.ppm 和 logo2.ppm，这两个文件名会在生成过程被用到，生成结束后会删除这两个文件。

4.5 制作根文件系统

拷贝软件包里 tools/mkyaffs2image 到变量 PATH 中的任一目录下，如/usr/local/bin

```
$ sudo cp Helper2416/tools/mkyaffs2image /usr/local/bin
```

解压 Helper2416/root-qtopia.tar.xz

```
$ sudo tar -xvf Helper2416/root-qtopia.tar.xz -C ~/workspace
$ cd ~/workspace
$ sudo chown fedora:fedora root-qtopia -R
```

生成 yaffs 文件系统

```
$ sudo mkyaffs2image 0 root-qtopia root-qtopia.img
```

注意：请带上 sudo，以保证 root-qtopia 里所有的文件都能被包含进去，mkyaffs2image 后面需要带一个 0 参数，缺少可能会出错。详情请看：

```
$ mkyaffs2image --help
```

让其他程序集成到根文件系统中

比如有一个交叉编译过的可执行文件 hello，将它拷贝到解压后的 target/usr/local/bin 中

然后执行

```
$ sudo mkyaffs2image 0 root-qtopia root-qtopia.img
```

注意：修改过的根文件系统与软件包中的 root-qtopia.img 大小可能会不一样了，如果要让 SD 卡能自动烧录，必须修改 u-boot 源码的 include/configs/smdk2416.h 文件中的宏 CONFIG_BOOTCOMMAND，具体参见 4.1 节编译 u-boot。

当然，以上制作根文件系统的方法是最简单的方法，如果想自己从头制作根文件系统，则需要自己编译 busybox，拷贝所需要的库等生成自己想要的根文件系统，相关资料网上一大把，这里就不作介绍了。

4.6 编译 Qtopia2.2

从 nokia 网站下的 qtopia2.2 是比较早期的编译器编译的，现在的编译器已经不能直接编译通过了，我们使用的 qtopia2.2 来自友善之臂（嵌入式 LINUX 前辈）的官方网站，该公司已经做好了编译补丁，确保能编译通过。编译方法很简单，解压缩后，直接运行目录下 build 脚本即可完成编译，如下：

```
$ tar xvf Helper2416/sources/arm-qtopia.tar -C ~/workspace
$ cd ~/workspace/arm-qtopia
$ ./build
```

最后生成的 qtopia 软件包在目录 qtopia-2.2.0/qtopia/image/opt 下，拷贝到目标板根目录即可。

本来，编译 qtopia 之前，需要编译数个关联软件包，但是，由于我们提供的编译器里已经集成了这些关联包，所以就少了很多麻烦，如果使用其它公司提供的编译器，则可能需要自己编译如下关联包：

libpng-1.2.23.tar.gz

zlib-1.2.3.tar.gz

e2fsprogs-1.41.14.tar.gz

tslib-1.4.tar

jpegsrc.v6b.tar.gz

另外，以上的所有软件包我们都已经为您准备好了，就在 Helper2416/sources/depends 目录下边，如果您感兴趣，可以看接下来的介绍。

下边介绍关联软件包的编译方法，下列方法会将关联文件集成到编译器中，下边配置中的—prefix 代表的就是编译器的库文件所在目录。

4.6.1 交叉编译 jpeg

```
$ tar -xvf Helper2416/sources/depends/jpegsrc.v6b.tar.gz -C /home/fedora/workspace/
$   cd ~/workspace/jpeg-6b/
$ ./configure --enable-shared --enable-static --prefix=/opt/toolchains/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot/ --build=i386 --host=arm-linux
```

然后修改生成的 Makefile，如下

```
CC= gcc 改为: CC= arm-linux-gcc
AR= ar rc 改为: AR= arm-linux-ar rc
AR2= ranlib 改为: AR2= arm-linux-ranlib
$ make
$ sudo make install-lib
```

4.6.2 交叉编译 e2fsprogs

```
$ tar -xvf Helper2416/sources/depends/ e2fsprogs-1.41.14.tar.gz -C ~/workspace/
$   cd e2fsprogs-1.40.2/
$   mkdir build
$   cd build
$ ./configure CC=arm-linux-gcc -enable-elf-shlibs --host=arm-linux --prefix=/opt/toolchains/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot/
$   make
$   sudo make install-libs
```

4.6.3 交叉编译 png

```
$ tar -xvf Helper2416/sources/depends/libpng-1.2.23.tar.gz -C   ~/workspace/
$ cd ~/workspace/libpng-1.2.23/
$ cp scripts/makefile.linux ./Makefile
```

修改 Makefile

将

```
CC= gcc
```

改为：

```
CC= arm-linux-gcc
```

将

```
AR_RC= ar rc
```

改为：

```
AR_RC= arm-linux-ar rc
```

将

```
RANLIB= ranlib
```

改为：

```
RANLIB= arm-linux-ranlib
```

将

```
prefix=/usr/local
```

改为：

```
prefix=/opt/toolchains/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot/
$ make
$ sudo -s
$ make install
```

注意：有的时候直接 sudo make install 会出错，可能是因为环境变量被重置的原因，sudo -s 则可以保持环境变量。

4.6.4 交叉编译 zlib

```
$ tar -xvf Helper2416/sources/depends/zlib-1.2.3.tar.gz -C ~/workspace/
$ cd zlib-1.2.3/
$ ./configure --shared --prefix=/opt/toolchains/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot/
```

编辑 Makefile

在 gcc 前都加上 arm-linux-，如下为修改过的

```
CC=arm-linux-gcc
LDSHARED=arm-linux-gcc -shared -Wl，-soname，libz.so.1
CPP=arm-linux-gcc -E
AR=arm-linux-ar rc
RANLIB=arm-linux-ranlib
$ make
$ sudo make install
```

4.6.5 交叉编译 tslib

```
$ tar -xvf Helper2416/sources/depends/tslib.tar -C ~/workspace/
$ cd  ~/workspace/tslib-1.3/
$ ./autogen.sh
$ ./configure  --host=arm-linux  --cache-file=arm-linux.cache --prefix=/opt/toolchains/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot/
$ make
$ sudo make install
```

需要注意的是，qtopia 校准屏时需要用到编译生成的 ts_calibrate，而且要将其拷贝到目标板的 /usr/local/bin/目录，否则新编译的 tslib 可能不能校准。

关于/usr/local/etc/ts.conf 的说明：

以下是文件内容：

```
module_raw input
#module pthres pmin=1
module variance delta=30
module dejitter delta=100
module linear
```

以上各模块是从下到上调用，但首先生效的模块是 input，依次回调类推。

module_raw input

表示直接从 input 设备输入

module pthres pmin=1

压力控制模块，最小压力为 1 时有效，对于我们用的电阻触摸屏，没有压力级别，值总是 1，所以该模块其实没用，因为其它很多开发板公司都有加上该模块，所以在此处说明。

module variance delta=30

这是消除异常模块，delta 的值两相邻点的原始坐标的平方和，即$(X_1-X_2)^2+(Y_1-Y_2)^2$，此处表示 delta 值大于 30 时可能为异常点。

module dejitter delta=100

这是消除抖动模块

module linear

这是用于坐标变换的模块，会自动调整到适合屏幕的分辨率。

4.6.6 交叉编译 qtopia

前边已经讲过了，如下步骤执行：

```
$ tar xvf Helper2416/sources/arm-qtopia.tar -C ~/workspace
$ cd ~/workspace/arm-qtopia
$ ./build
```

执行完上边步骤后，qtopia 会安装在编译目录下的~/workspace/arm-qtopia/qtopia-2.2.0/qtopia/image/opt/Qtopia 目录下，如果要将开发板上的 qtopia 文件替换掉，可先删除或者更名开发板上原有的 qtopia，只要删除/opt 目录就可以，然后把编译生成的 qtopia/image/opt 整个拷贝到根文件系统上，还要把前边交叉编译生成的各个动态库也拷到根文件系统下的 lib 里（当然，我们提供的根文件系统里已经包含了这边库文件）。

比如你原来根文件系统目录为 target，是通过 nfs 加载启动（参见 6.5 nfs 一节）的，可用以下命令替换 qtopia

先从原根文件系统目录下拷贝字体到 target/opt/Qtopia/lib/fonts/下

```
$ cp -r target/opt/Qtopia/lib/fonts/* ~/workspace/qtopia/image/opt/Qtopia/lib/fonts/
```

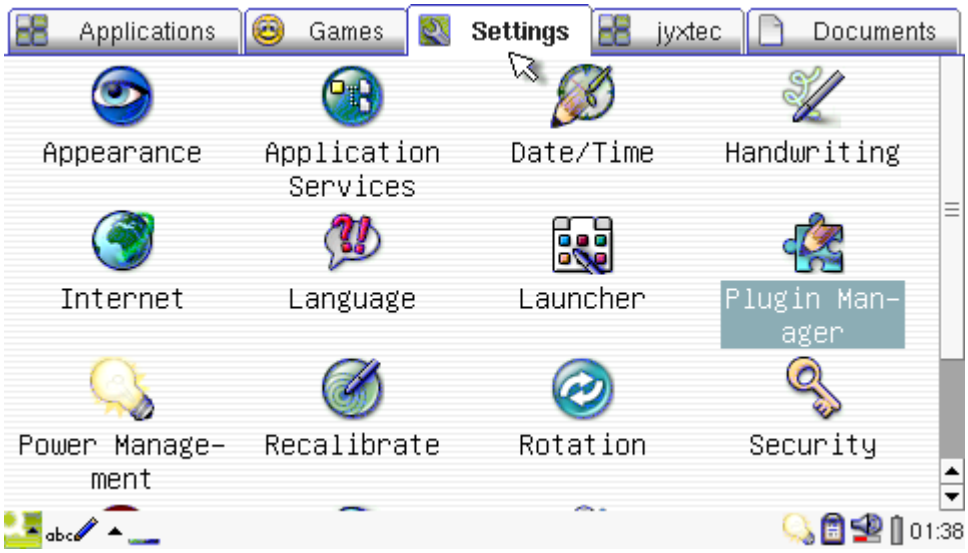
删除旧的

```
$ rm   target/opt   -rf
```

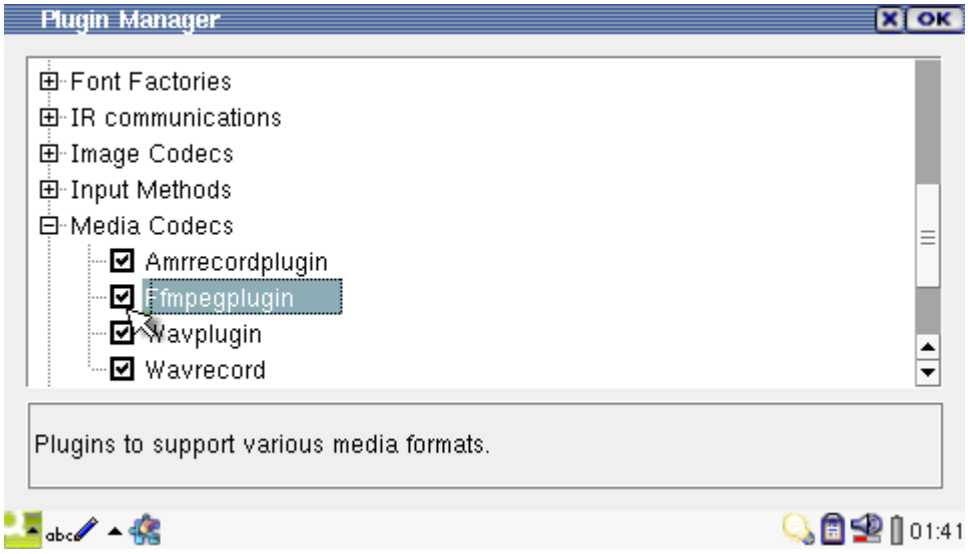
拷贝新的

```
$cp   -rf   ~/workspace/qtopia/image/opt    target/
```

然后就可重启板子了，启动起来后，要根据需要手动使能插件功能，否则可能音、视频不能播放，点击 Settings->Plugin Manager，如下图：



勾上你想要支持的功能，如要播放视频，则需要勾上 Media Codecs->Ffmpegplugin，要录音回放则要勾上 Wavplugin 和 Wavrecord，如果不知道怎么选，先全部勾上就对了。



此后，您就随便玩吧。

4.7 编译 qt4

软件包里 Helper2416/source/qt4.7.2.tar.gz 是 qt4.7.2 的源码，解压缩 qt4.7.2 的源码到 workspace 目录：

```
$ tar -xvf qt4.7.2.tar.gz -C ~/workspace
```

为了方便编译，我们特地制作了编译脚本，直接运行脚本即可完成编译(此脚本针对 qt4.7.2,其他版本可自行修改脚本)，qt4 也会依赖 tslib，在我们的编译器里已经集成，请确保正确安装交叉编译器(参照 3.2 安装交叉编译器一节)。进入 qt4.7.2 目录开始编译：

```
$ cd ~/workspace/qt4.7.2
$ ./build_all_qt4.7.2
```

编译开始后，且需要相当长的一段时间才能结束，编译脚本的最后会用到了 sudo，因此需要用户在编译时输入 sudo 的密码，最后会把 qt4 安装到/usr/local/Trolltech/QtEmbedded-4.7.2-arm 目录下,执行 mk-qt4-target,把需要拷贝到目标板的根文件系统的文件打包起来

```
$ ./mk_qt4_target
```

拷贝 switch_to_qt4 到目标板的根文件系统下(假设根文件系统在~/workspace/target 下)

```
$ cp switch_to_qt4 target/
```

然后安装 qt4 到根文件系统中

```
$ cp target-qte-4.7.2.tgz target
$ cd target/
$ tar -xvf target-qte-4.7.2.tgz
```

拷贝 switch_to_qt4 到根文件系统下

```
$ cp switch_to_qt4
```

打开超级终端，启动板子，设置 qt4 的环境

```
$ source ./switch_to_qt4
```

运行一个 qt4 的例子程序

```
$ /usr/local/Trolltech/QtEmbedded-4.7.2-arm/examples/widgets/wiggly/wiggly -qws
```

如果要重新运行 qtopia，可以关闭 qt4 程序，然后在终端中运行

```
$ /bin/qtopia
```

4.8 编写 Hello World 程序

熟悉编程的朋友对 hello world 程序再熟悉不过了，我们不多讲，下边介绍编译方法。

使用文本编辑器新建 hello.c 并写入以下代码：

```
#include <stdio.h>

int main(void)
```

```
{
    printf("Hello，World!\n");
}
```

编译生成板子可执行的文件 hello

```
$ arm-linux-gcc hello.c -o hello
```

接下来只要把 hello 传到开发板上就可以在板子上执行它了，方式有多种：

放到根文件系统中，做成根文件系统镜像烧进 nandflash 中；

放到 NFS 加载的根文件系统目录下，板子从 NFS 启动；

使用 U 盘或 SD 卡；

使用串口；

注：生成的文件传到开发板上时，要注意是否有执行属性，如果没有则需要输入如下类似的命令增加执行属性才能执行(否则会出现 Permission denied 的错误提示)：

```
$ chmod +x hello
```

执行程序：

```
$ ./hello
```

就会打印出：

```
$ Hello，World!
```

5 传输文件到目标板

5.1 使用 U 盘传输文件

U 盘插入 PC 机

打开 vmware 虚拟机，鼠标放在窗口右下角位置，每个图标会显示相应的信息，找到相应的 U 盘图标，可能是 Mass Storage Device，点击出现如下菜单，

选择 Connect (Disconnect from Host)，之则 U 盘会从 windows 里消失，然后被挂到 fedora 系统里，此时，已经可以直接拷贝文件到 U 盘里了。

说明：如果插入 U 盘的时候，正好是在 vmware 窗口的话，U 盘会自动挂载到 fedora 里。

5.2 使用 SD 卡传输文件

与 U 盘差不多，挂载到虚拟机，拷贝 hello 到 SD 卡，安全卸载后插入板子，手动 mount 到如/mnt 目录，就可在/mnt 下执行 hello， 或者再拷贝文件到板载 NAND Flash

```
$ mount -t vfat /dev/mmcblk0p1   /mnt
$ ./hello
```

5.3 使用串口传输文件

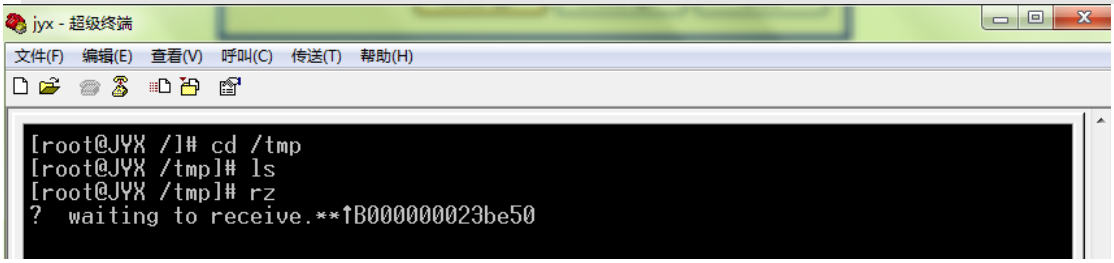
向开发板发送文件

打开超级终端，如要传输文件到/tmp 则进入到/tmp

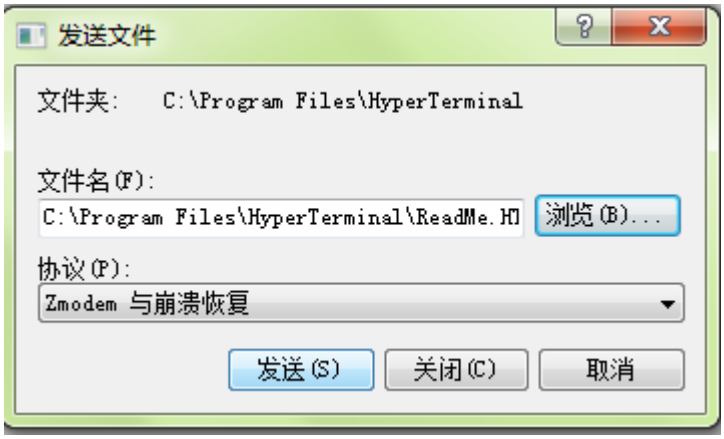
```
$ cd   /tmp
```

然后输入命令，从 pc 机接收文件：

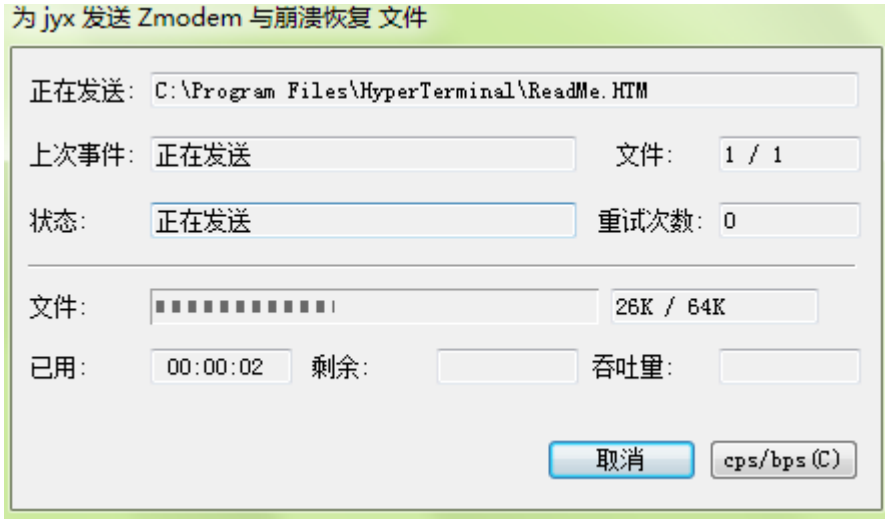
```
$ rz
```



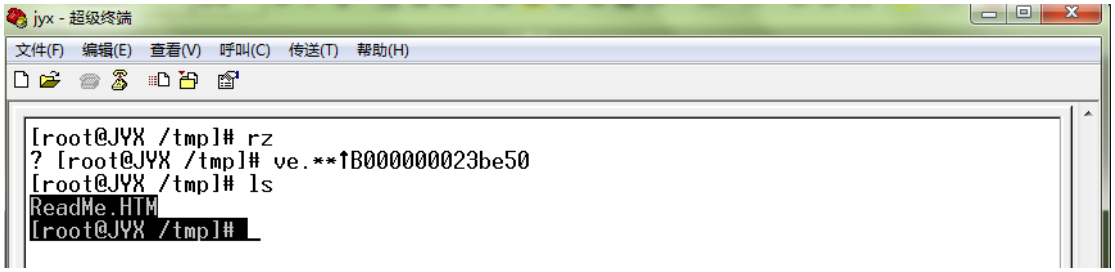
在终端中右键选择‘‘发送文件’’，浏览按键选择要传输的文件如 ReadMe.HTM，并设置好协议：



点发送

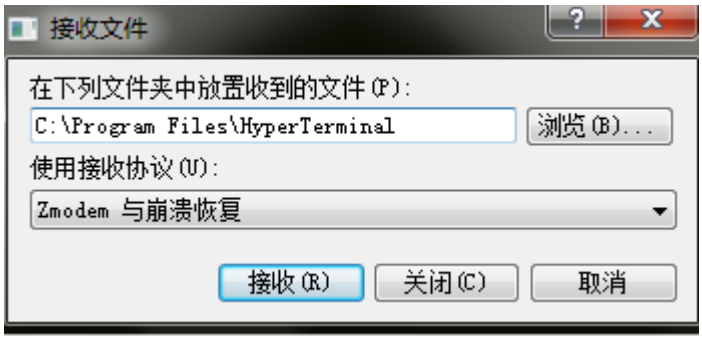


发送完成后可以用 ls 查看/tmp 目录，可以看到目录下多了 ReadMe.HTM，就是刚刚传输的文件



向 PC 机发送文件

超级终端中，点右键，选择“接收文件”，设置好接收目录与协议



然后点“关闭”，并在终端中输入“sz /test.txt”，便会向 PC 传送根目录下的 test.txt 文件

5.4 使用 usb-gadget 方式传输文件

Helper2416 支持 usb-gadget，可以把开发板上的存贮设备模拟成 U 盘。可以直接从电脑将文件拷贝进去。在 4.3 节我们已经介绍了如何编译模块，要将开发板当 U 盘用就会用到 g_file_storage.ko 模块文件，在我们的根文件系统里已经提供了该模块。下面介绍将开发板上的 SD 卡映射成电脑上的 U 盘：

1、将 SD 卡插入 SD1 卡座

2、插上 USB 线，连接电脑和开发板的 MicroUSB 接口

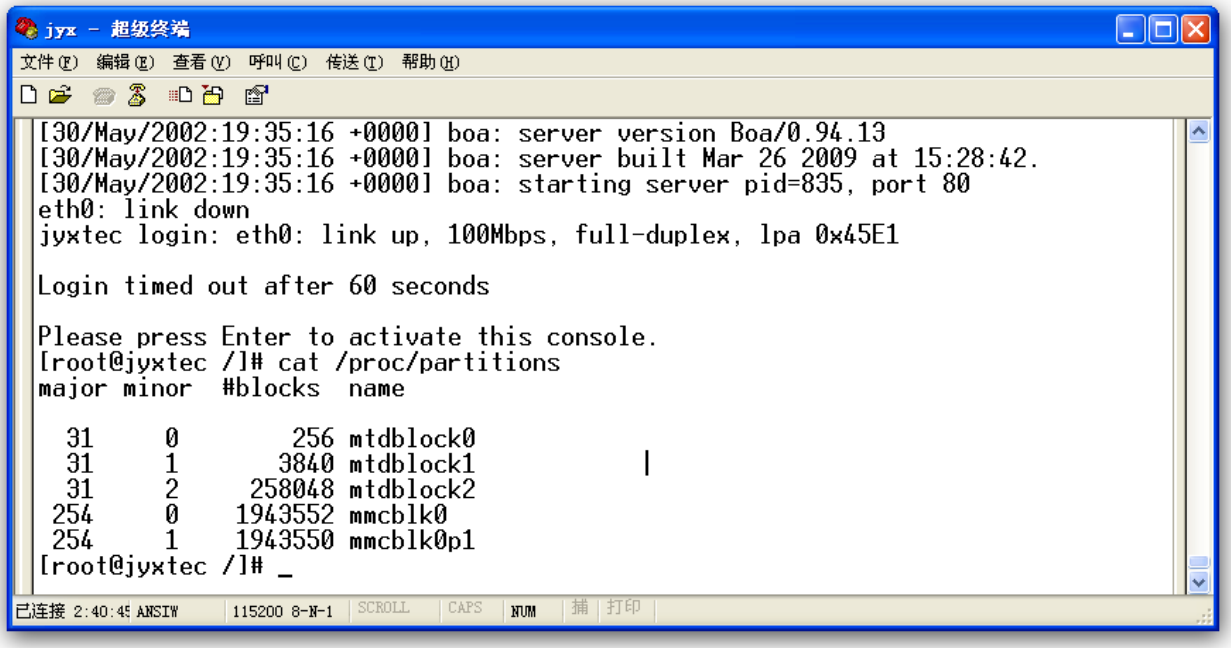
3、开机，并等待启动完毕

4、登录串口终端

5、查看 SD 卡所在的设备分区

```
cat /proc/partitions
```

如下图：



根据经验看到 SD 卡所在的分区是 mmcblk0p1，对应的设备是：/dev/mmcblk0p1

6、挂载 SD 卡设备：

```
$ insmod g_file_storage.ko file=/dev/mmcblk0p1 removable=1
```

这时电脑上就会出现一个 U 盘设备，格式化、拷备文件等 U 盘的操作，都可以做了。

说明：SD 卡、U 盘设备一般都是对应两个设备名，取后边一个，当然也有是一个设备的，叫做“Entire device “，但这种情况很少。

6 网络服务

6.1 配置开发板网络接口

查看网络接口配置信息

```
#ifconfig
```

网卡禁用，如果板子是从 nfs 启动，则不可将网卡禁用，否则系统无法正常工作

```
#ifconfig eth0 down
```

配置网卡的硬件地址

```
#ifconfig eth0 hw ether 08:90:90:90:90:20
```

配置 ip　子网掩码　并启用网卡

```
#ifconfig eth0 192.168.0.20 netmask 255.255.255.0 up
```

配置默认路由

```
#route add default gw 192.168.0.1
```

设置 DNS 服务器

```
#echo nameserver　202.82.1.1 > /etc/resolv.conf
```

在根文件系统中 etc/init.d/rcS 对板子网络接口进行了配置，如下：

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

第二行命令会根据/etc/eth0-setting 文件配置

其默认内容如下：

```
IP=192.168.0.20
Mask=255.255.255.0
Gateway=192.168.0.1
DNS=202.82.1.1
MAC=08:90:90:90:90:20
```

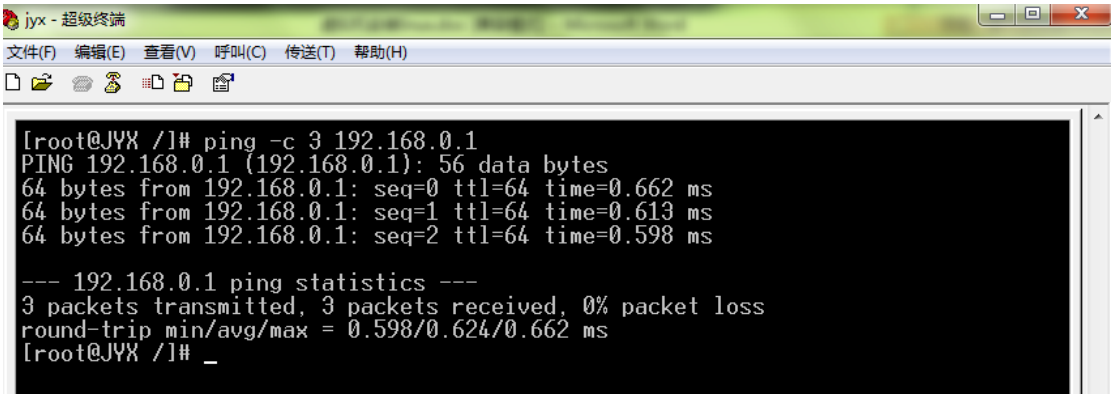
可根据需要修改。

检测网络状态

设置完网络接口，可用 ping 命令进行测试网络状态，

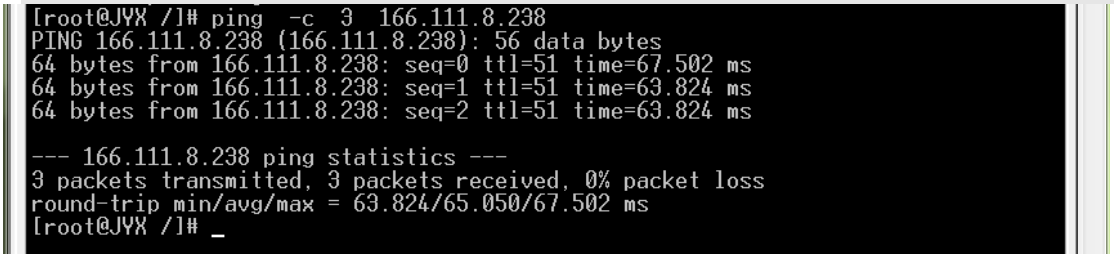
```
#ping　-c　3　192.168.0.1
```

如图所示表示网卡正常，可 ping 通网关，已连上局域网

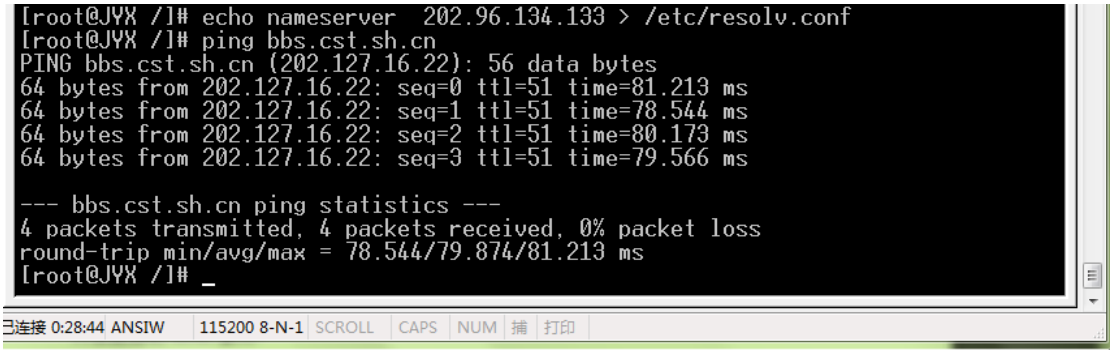


测试是否可连外网，则可 ping 外网 ip 如水木清华 BBS(ip 为 166.111.8.238)

```
#ping  -c  3  166.111.8.238
```



想直接 ping 外网网址，必须设置好 DNS(域名解析服务器)，查看您当前网络使用的 DNS，可询问网络管理员，或者网上搜索，如深圳电信 DNS 之 - 为 202.96.134.133



6.2 telnet

开发板启动 telnet 服务

```
#/etc/rc.d/init.d/netd start
```

默认这个服务是自动启动的，可查看/etc/init.d/rcS 文件，

若板子 ip 为 192.168.0.20，主机登录板子可在其终端下输入以下命令：

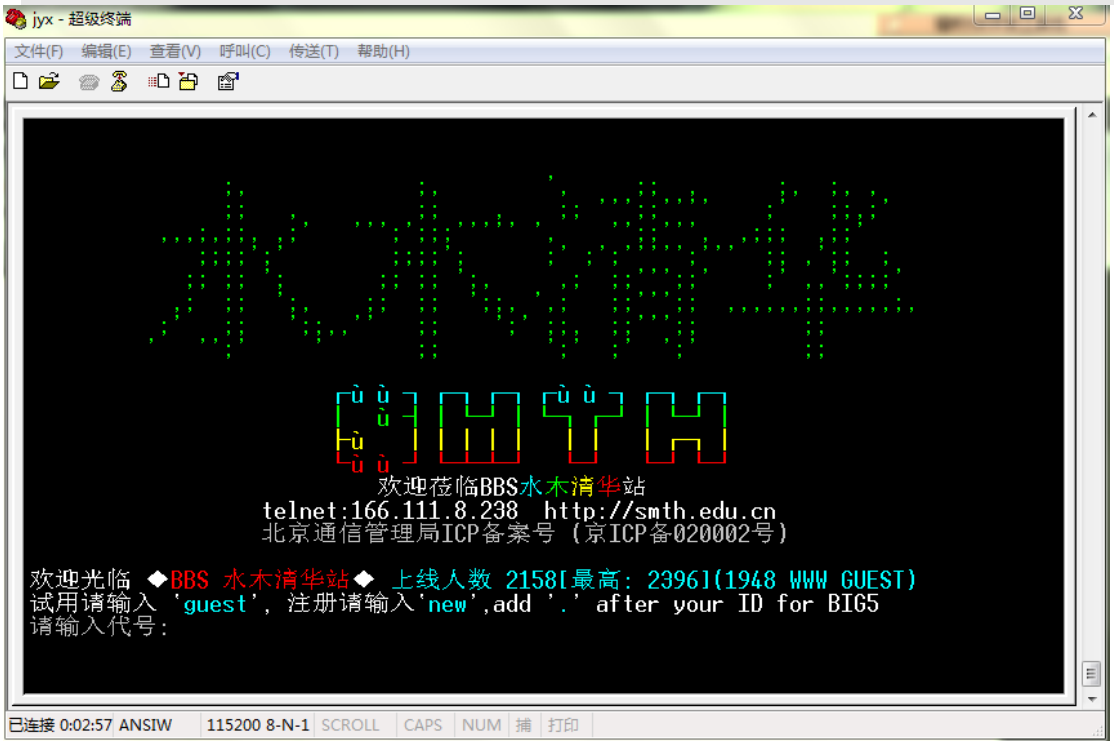
```
# telnet 192.168.0.20
```

然后输入 root 回车

要退出连接则输入 exit

也可从板子登录到其他提供了 telnet 服务的机器，用法是 telnet 后接服务器 ip，比如登录水木清华 BBS:

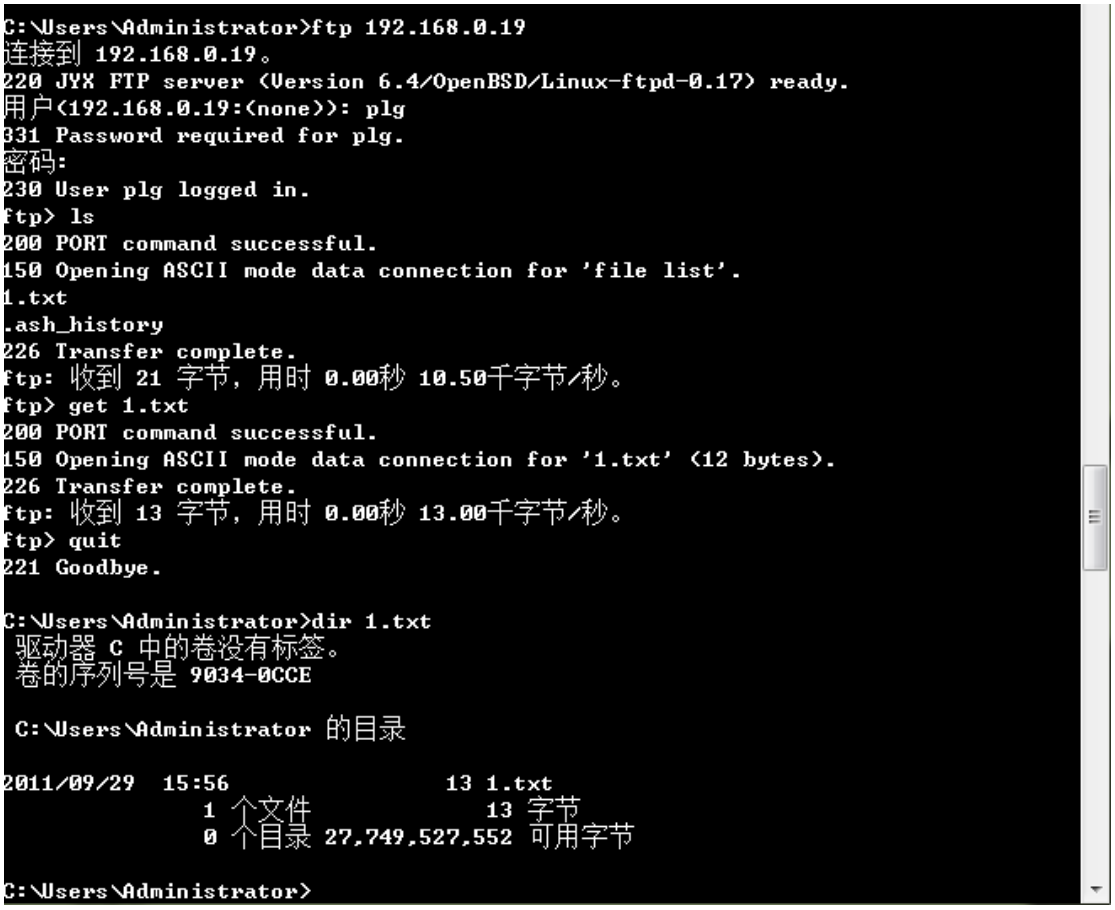
```
# telnet www.newsmth.net
```



6.3 ftp

板子提供了 ftp 与 ftpd， 可以用 ftp 登录远程主机或其他机子登录板子进行文件传输，板子提供了一个 ftp 帐户：plg 密码为:plg

如图是 windows 系统从板子下载/home/plg/下 1.txt 文件的例子



6.4 http

启动 http 服务

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

把网页放到/www/目录下

其他机子可通过浏览器中输入：http://192.168.0.20 来访问

6.5 nfs

安装配置 nfs 服务

启动虚拟机 linux(ubuntu)，如果未安装 nfs 服务相关软件则安装

```
$ sudo apt-get install nfs-kernel-server
```

使用网络文件系统之前，我们需要从软件包中解压根文件系统，保存到~/workspace/target 目录下，参考《根文件系统制作》一节

编辑器(vi 或 gedit)打开/etc/exports 并添加以下内容

```
/opt/target *(rw,sync,no_root_squash)
```

* 表示所有的客户机都可以挂载此目录

rw 表示可读写权限

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

启动 nfs 服务

虚拟机中执行以下命令

```
$ sudo service portmap start
$ sudo service nfs-kernel-server start
```

并关闭防火墙

```
$ sudo ufw disable
```

挂载

启动开发板，启动完成后，通过超级终端连接板子并输入以下命令(假设虚拟机 ip 为 192.168.0.100)

```
# mount -t nfs   -o nolock,tcp 192.168.0.100:/opt/target /mnt
```

如果要卸载，输入以下命令

```
#umount /mnt
```

板子通过 nfs 启动

重启板子，加载内核前按任意键进入 u-boot 命令行，输入以下命令设置启动参数

```
#set bootargs console=ttySAC0，115200 root=/dev/nfs mem=64m
nfsroot=192.168.0.100:/opt/target ,tcp rw ip=192.168.0.20 init=/linuxrc
```

保存并继续启动

```
#save
#boot
```

如果出现 nfs: server 192.168.0.100 not responding， still trying 指示，则可试着这样修改启动参数：

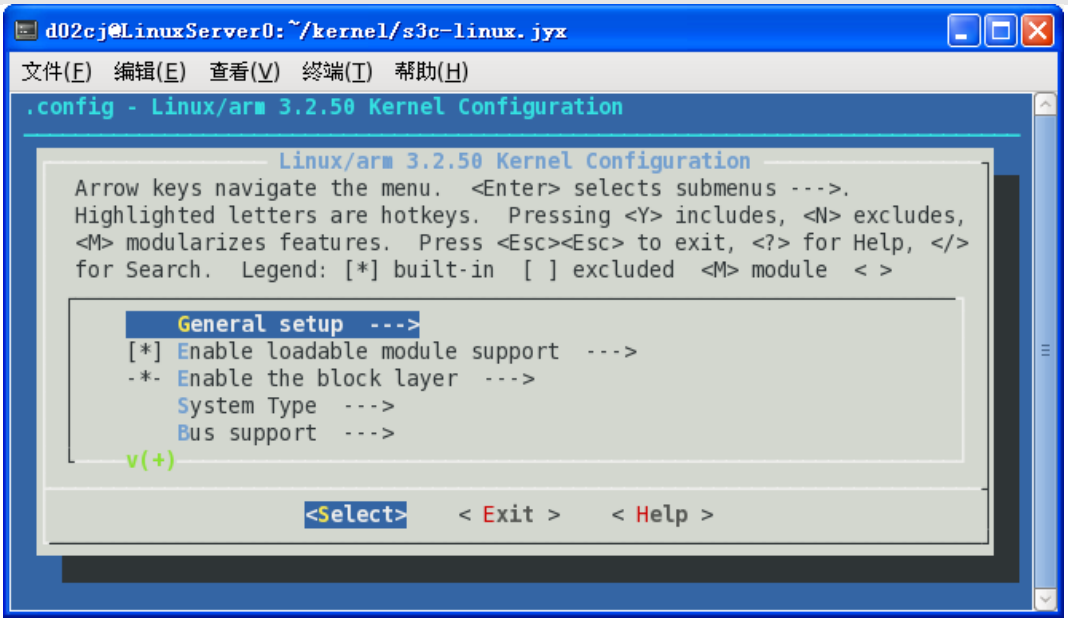
```
#set bootargs console=ttySAC0,115200 root=/dev/nfs mem=128m
 nfsroot=192.168.0.100:/opt/target,tcp,nolock,rsize=1024,wsize=1024 rw ip=192.168.0.20
 init=/linuxrc
```

7 内核配置

本章的所有内核配置都已经完成，下边的介绍只是让用户熟悉内核配置的过程。

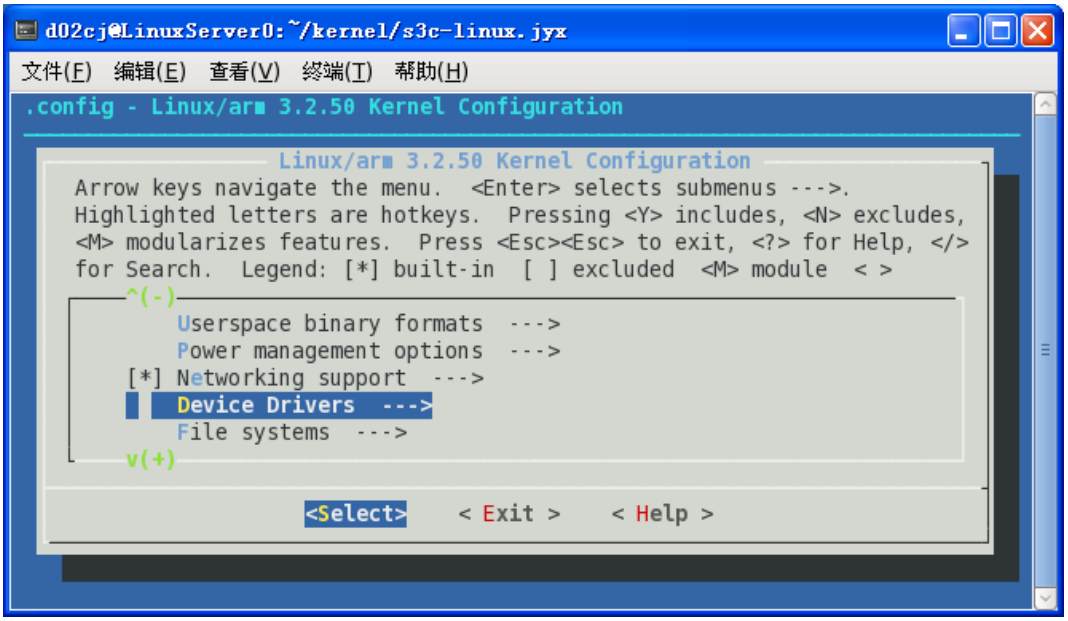
执行内核配置前可先加载默认的配置，再进行修改，以后修改的配置内容会保存到内核目录下的.config 文件中，输入如下命令，进入配置环境：

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

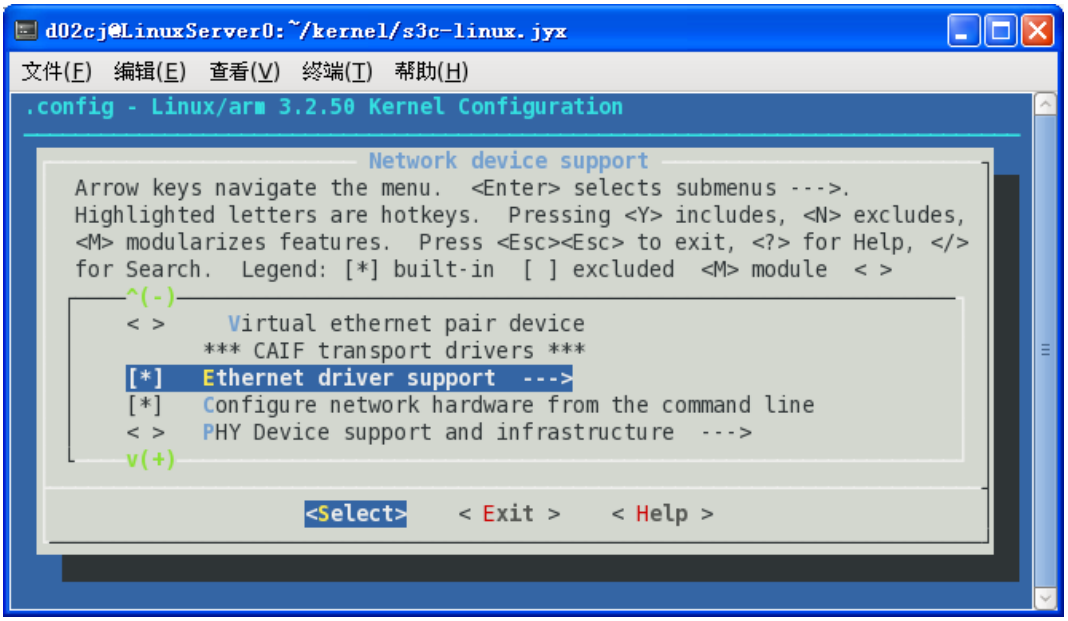


7.1 网卡驱动

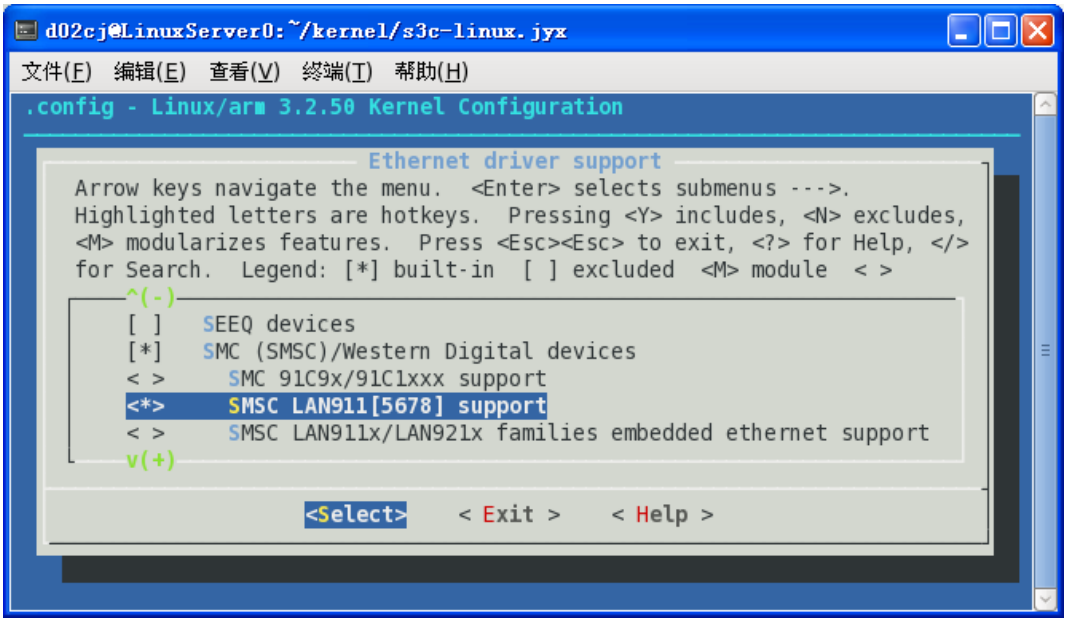
主菜单中，选择 Device Drivers



约在中间位置找到 Netwoking support

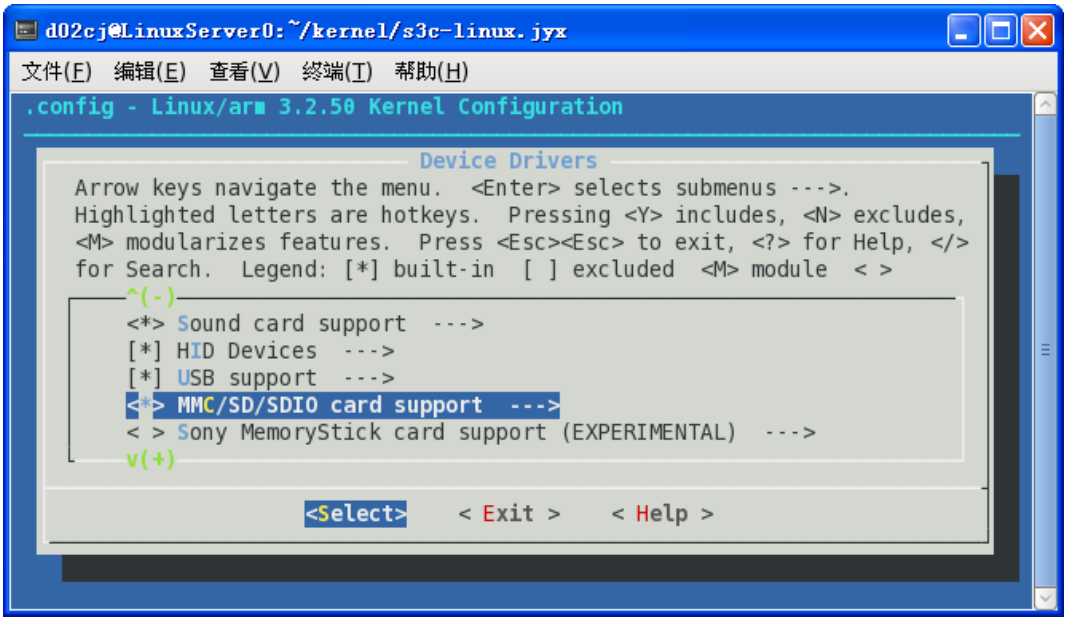


进入 Ethernet driver support， 如图选择支持 SMSC 网卡

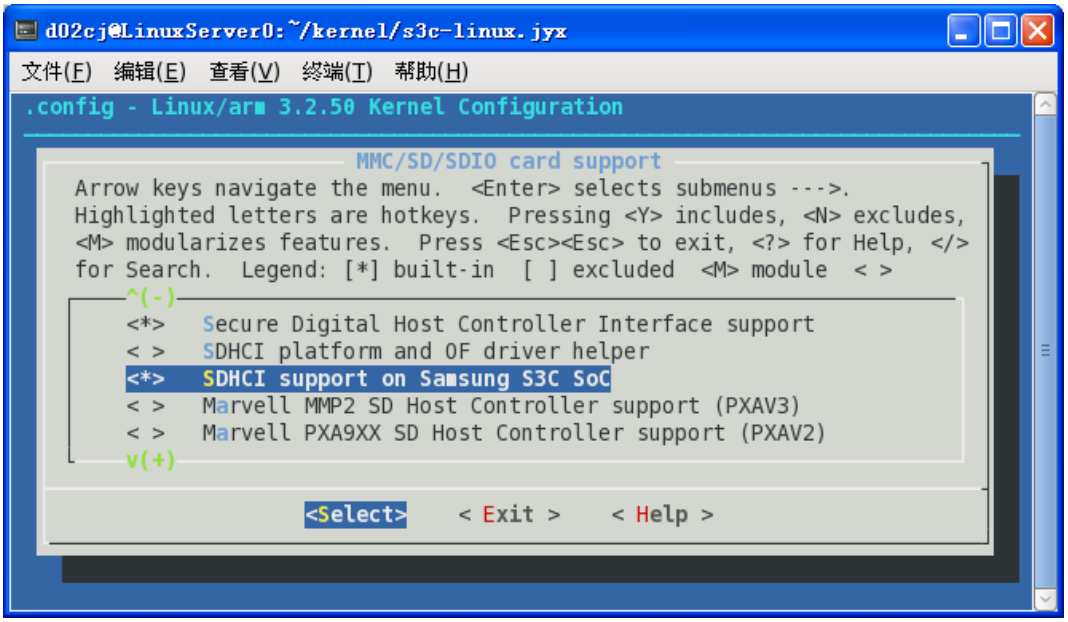


7.2 MMC 卡驱动

主菜单选择 Device Drivers 并拉到最下面选择 MMC/SD Card support

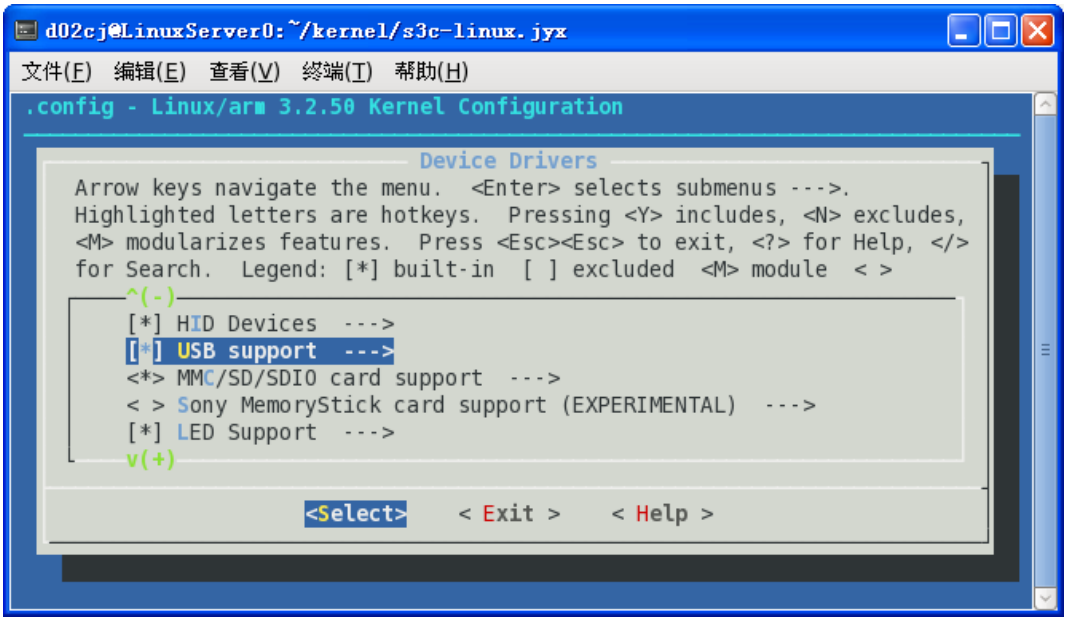


如图选择 SDHCI Supporton Samsung S3C Soc：

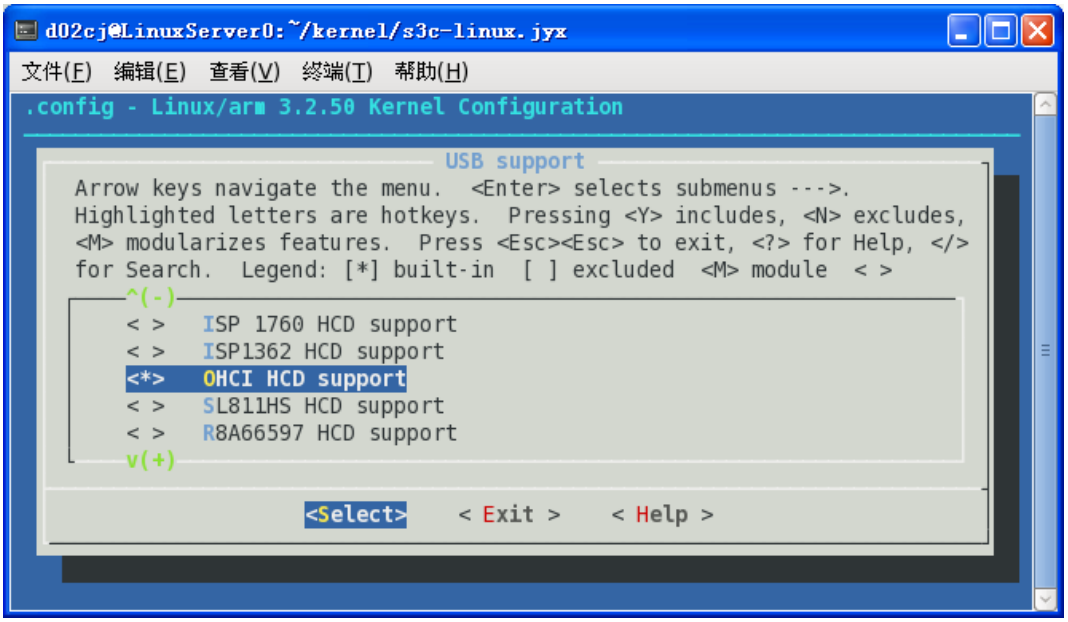


7.3 USB 驱动

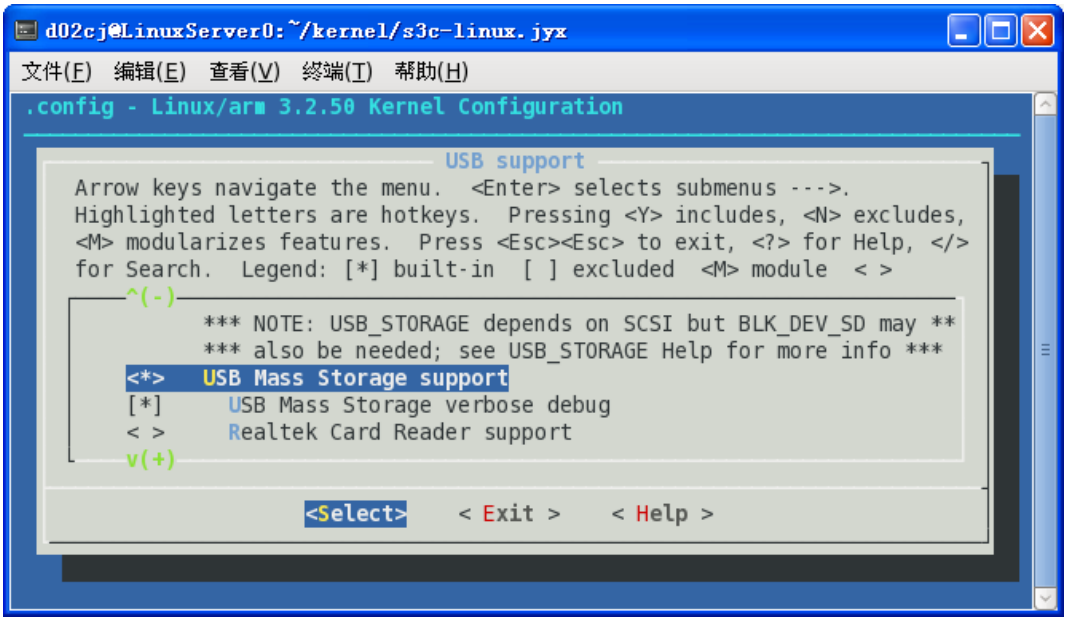
主界面选择 Device Drivers 并拉到最下面选择 USB support



如图选择，Support for Host-side USB 及 OHCI HCD Support，表示支持 USH Host 驱动。

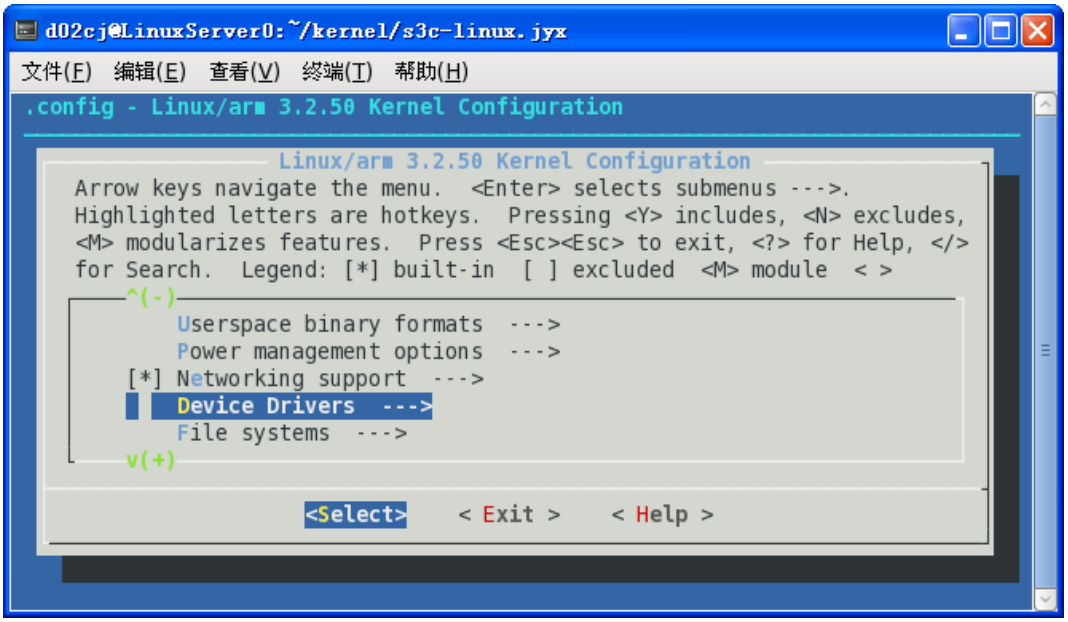


向下键找到并选择 USB Mass Storage support，以支持 U 盘等设备：

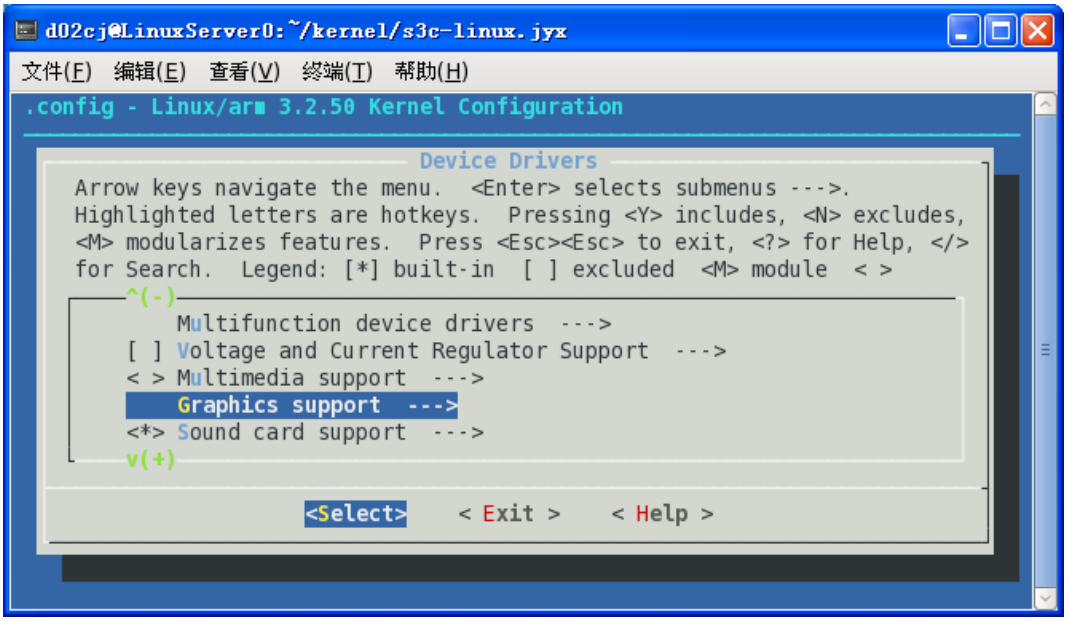


7.4 LCD 驱动

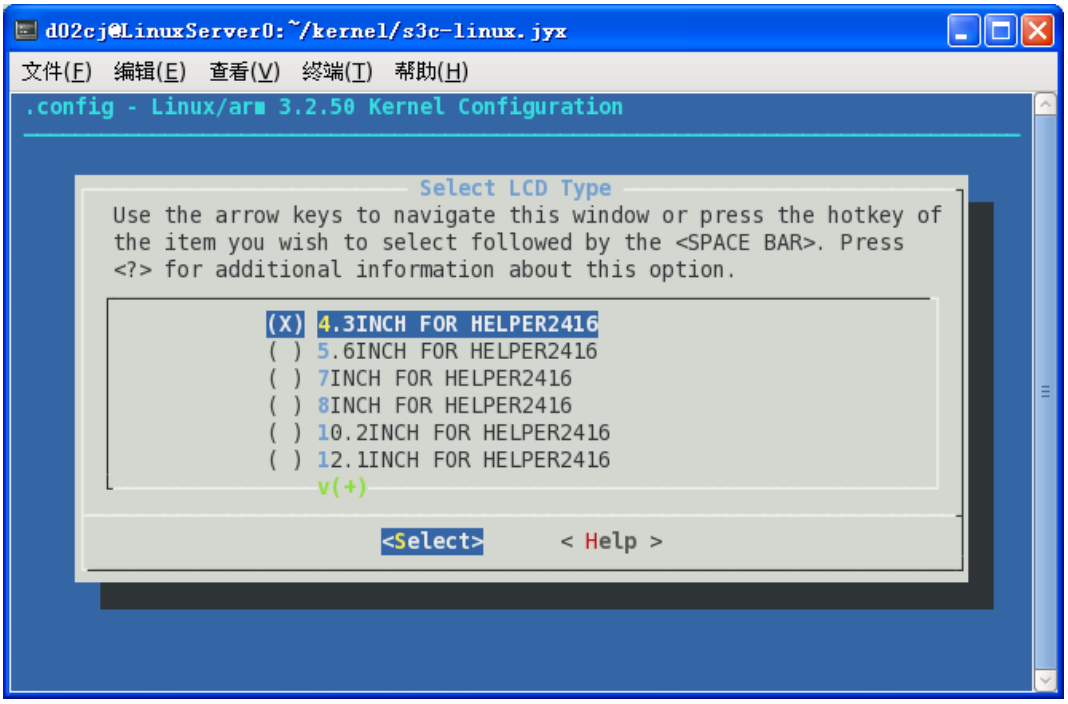
选择 Device Drivers



进入 Graphics support



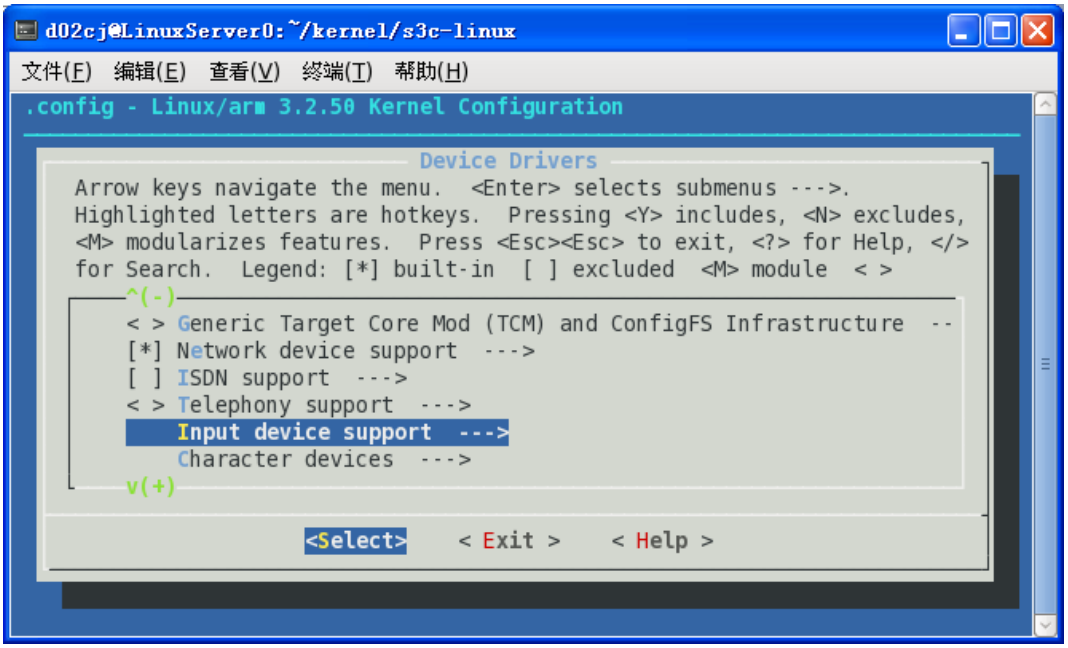
选中 Support for frame buffer devices，进入 Select LCD Type：



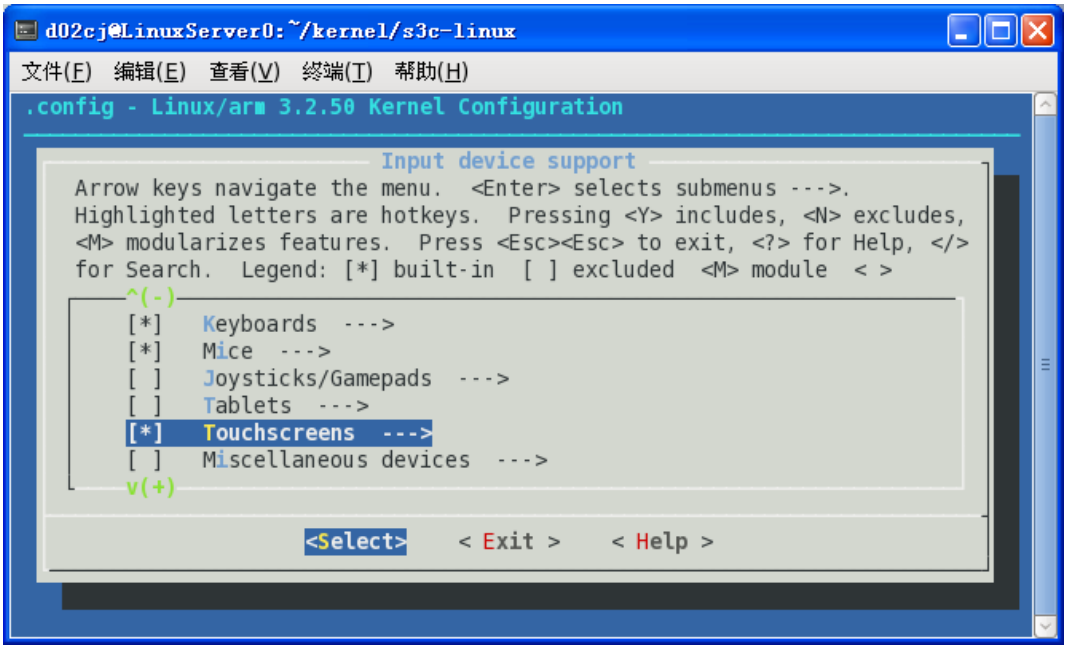
根据所用的液晶屏尺寸选择相应尺寸的 LCD 驱动。

7.5 触摸屏驱动

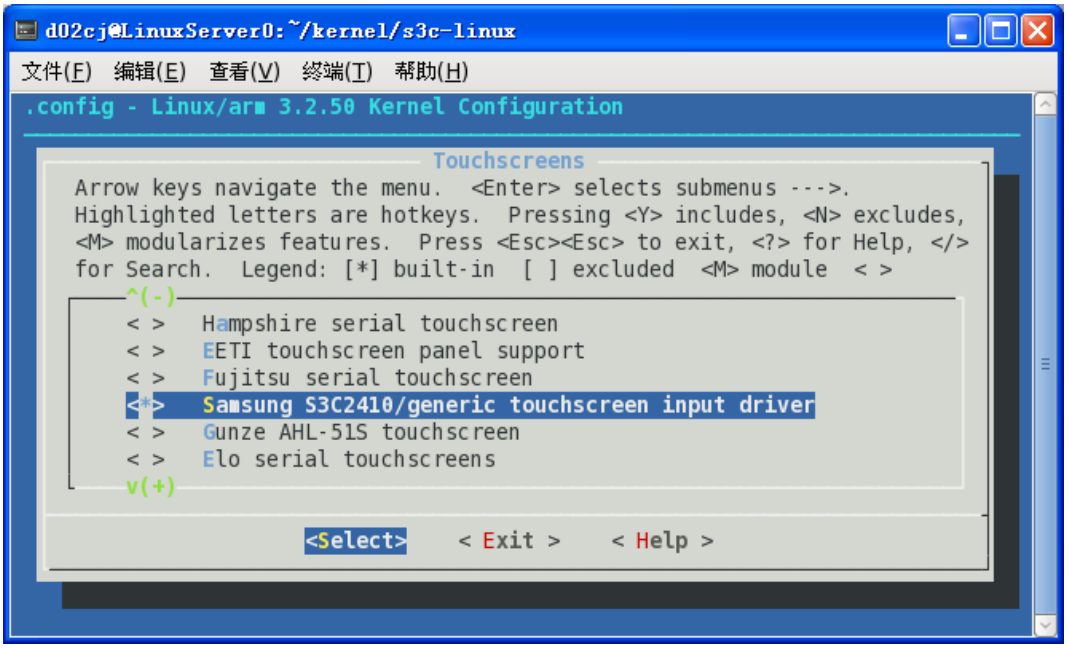
Device Drivers->Input device support 如下图：



选中 Touchscreens，同时回车进入

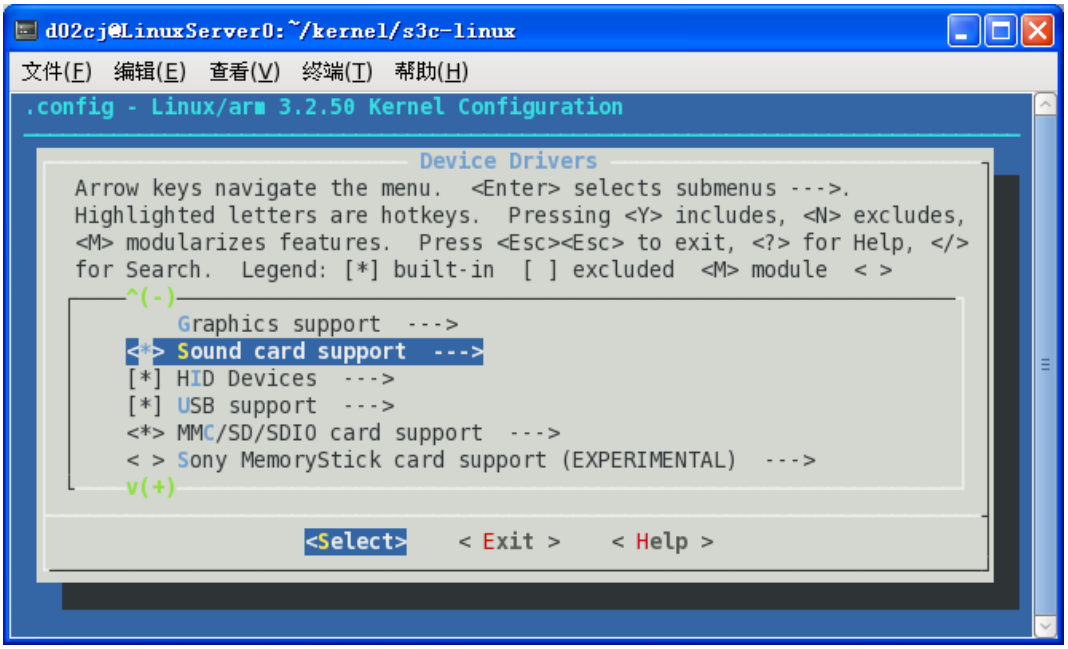


选中 Samsung S3C2410/generic touchscreen input driver

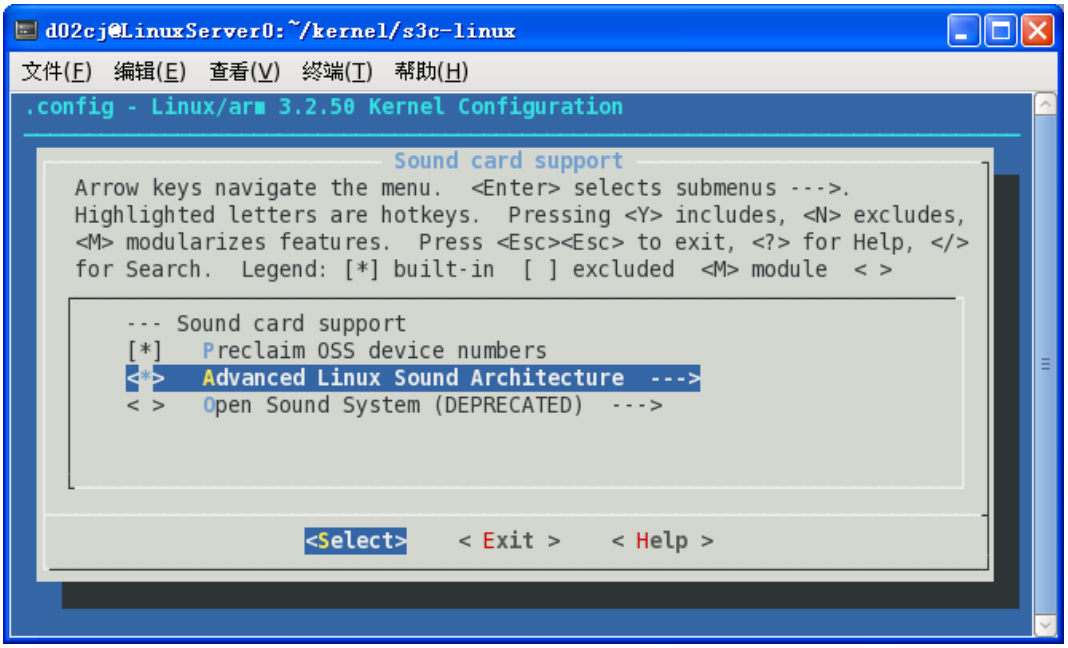


7.6 声卡驱动

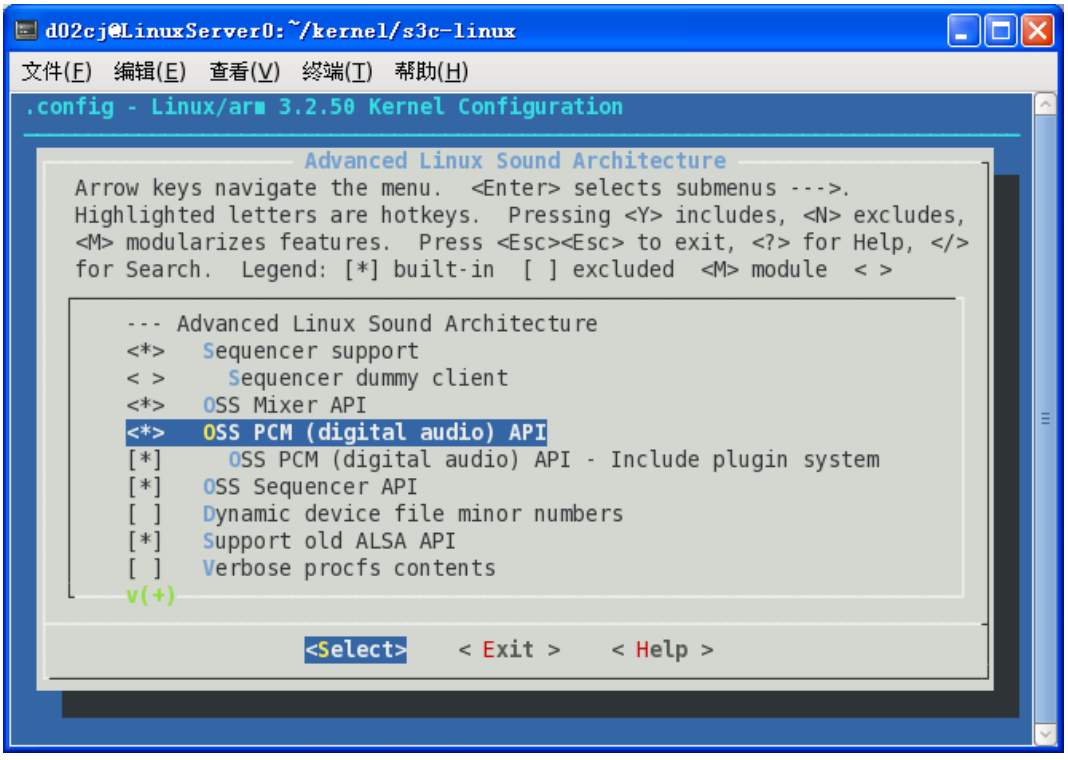
进入 Device->Sound card support



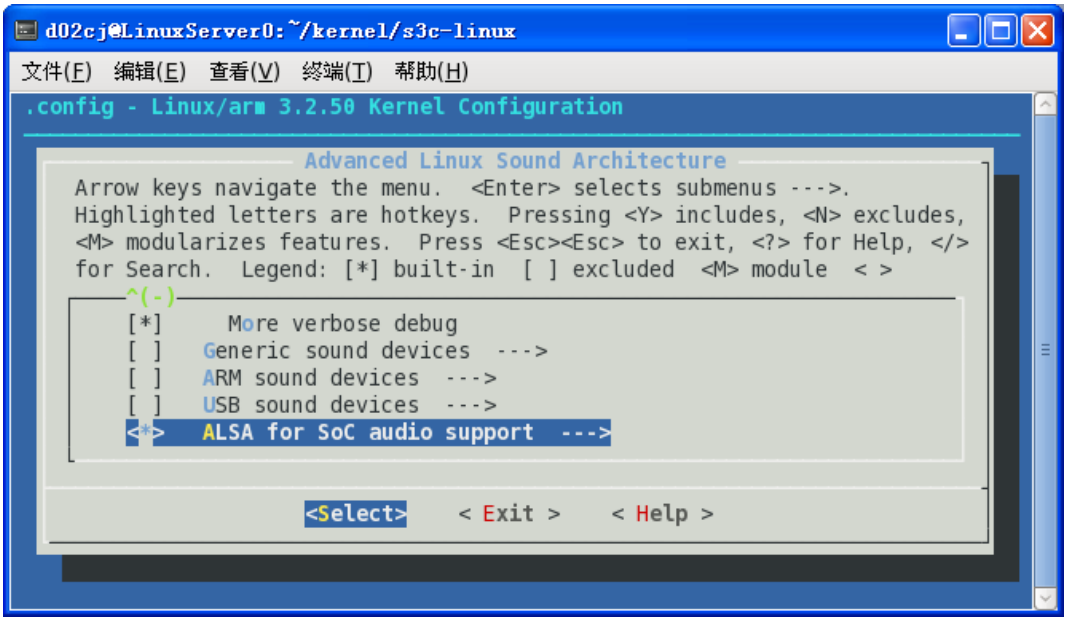
选择 Sound card support 后再进入 Advanced Linux Sound Architecture



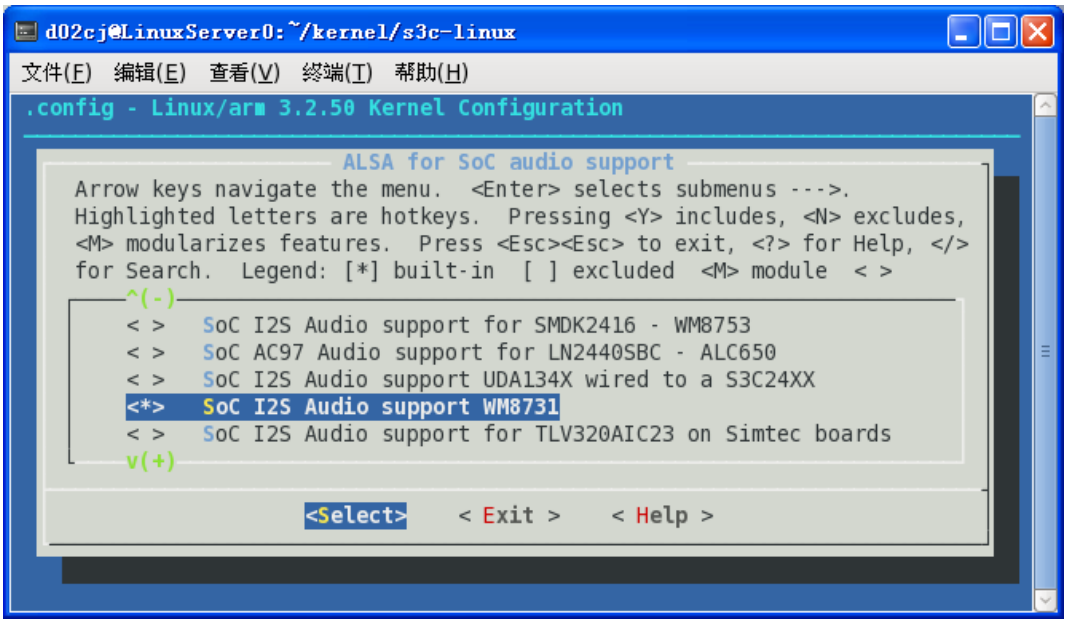
按下图选择



再拉到最下面选择 ALSA for SoC audio support

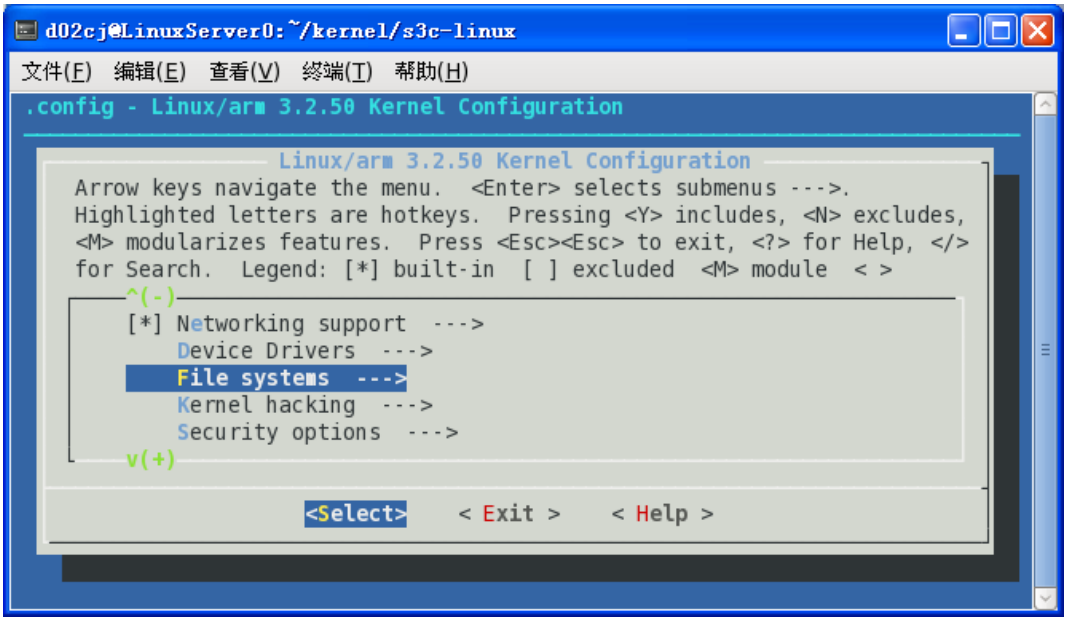


选择 ALSA for SoC audio support，进入并选中 ASoC support for Samsung 和 WM8731 和，如图所示：



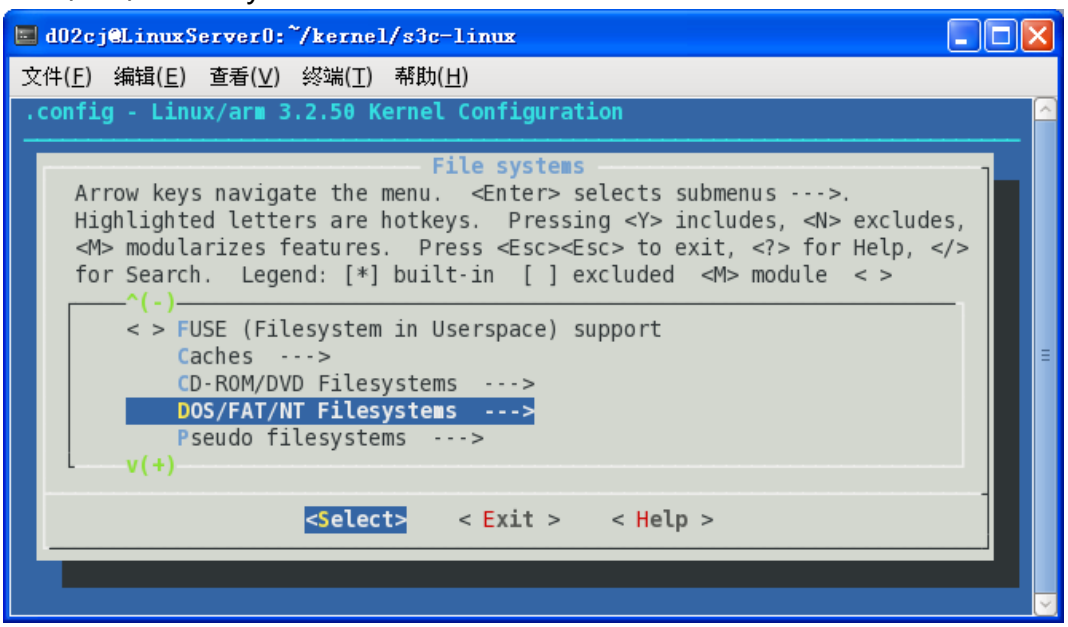
7.7 文件系统配置

选择 File systems

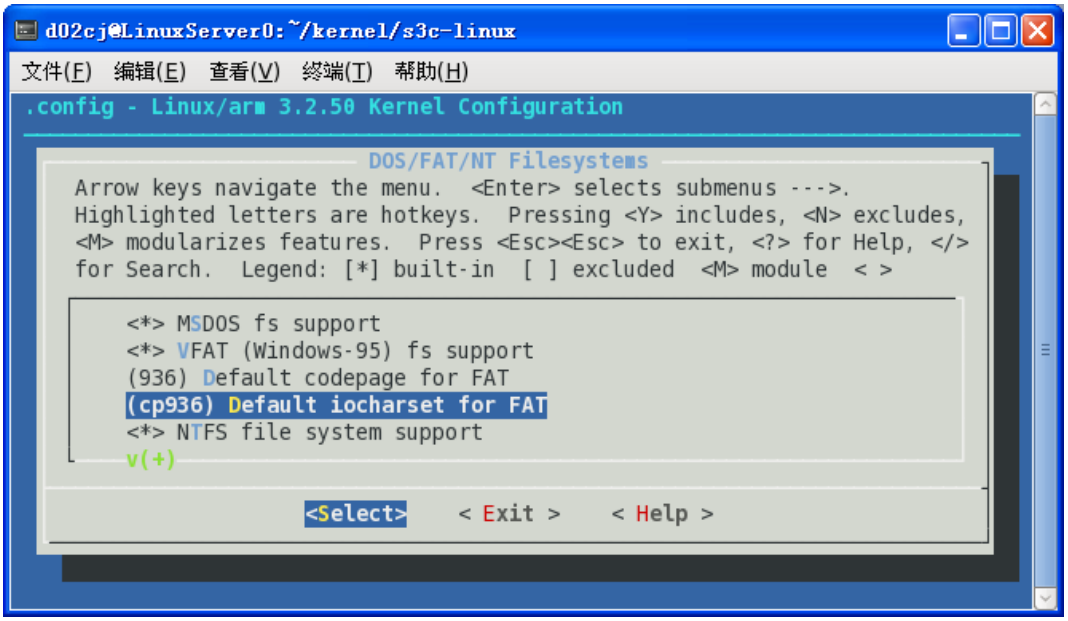


FAT32 支持

选择 DOS/FAT/NT Filesystems



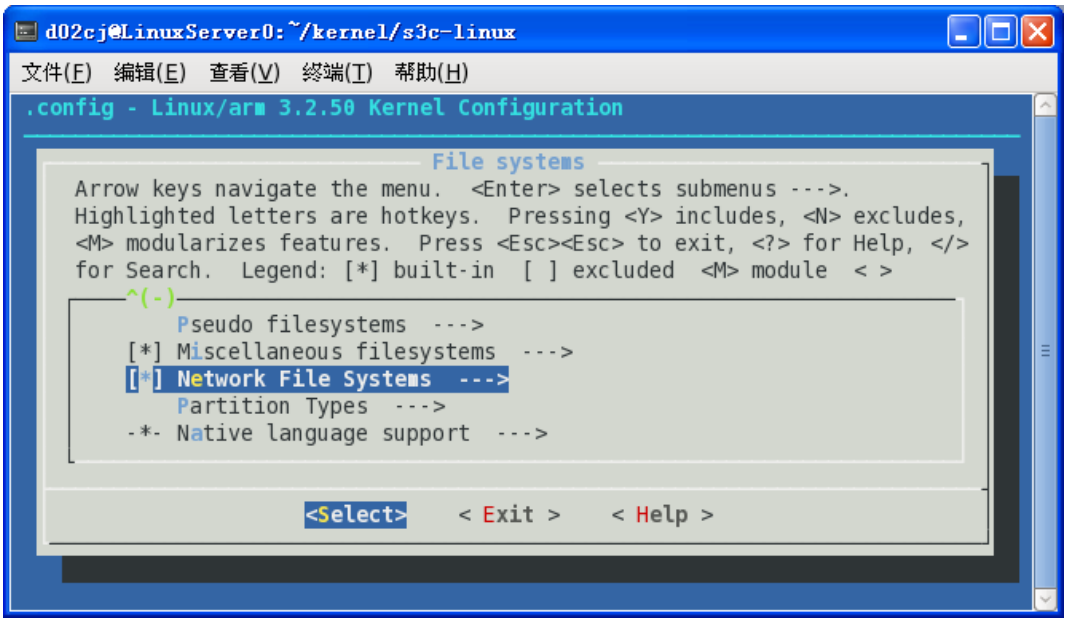
选择 VFAT，CP936 代表文件系统是简体中文字符集编码，



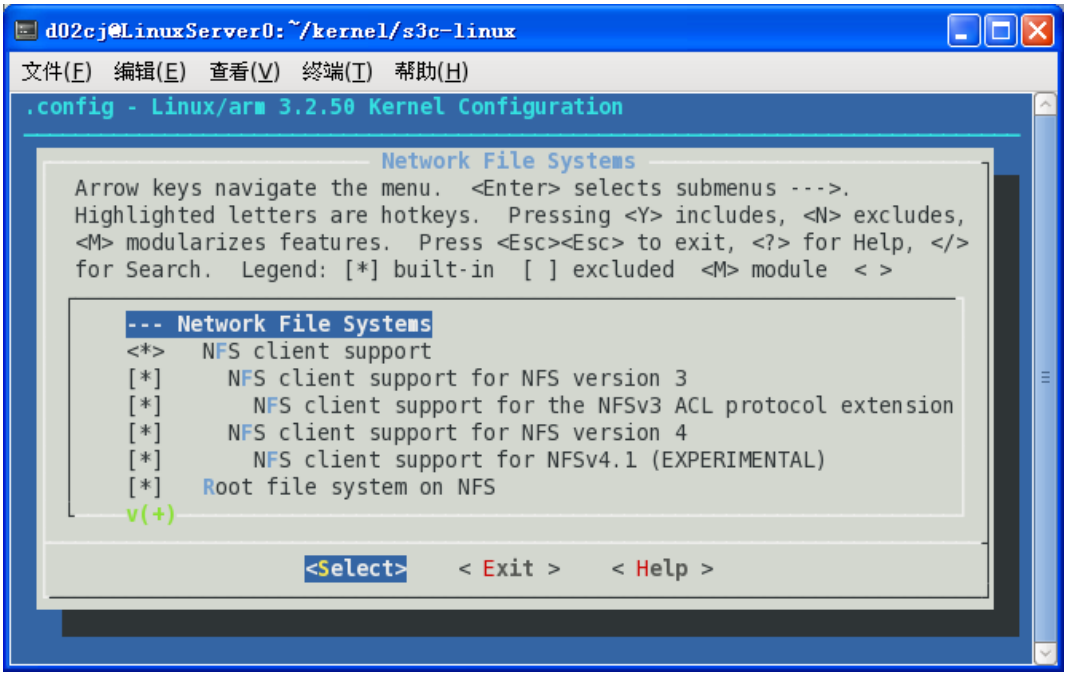
注：LINUX 支持很多文件系统，可根据需要选择，其中 NTFS 文件系统是有限支持，读没有问题，写不保证完全正确，所以如果想用 NTFS 文件系统，请谨慎选择。

NFS 支持（在开发的时候用，可用于加载文件系统，大大提高开发效率）

同上进入 File systems 选择 Network File Systems

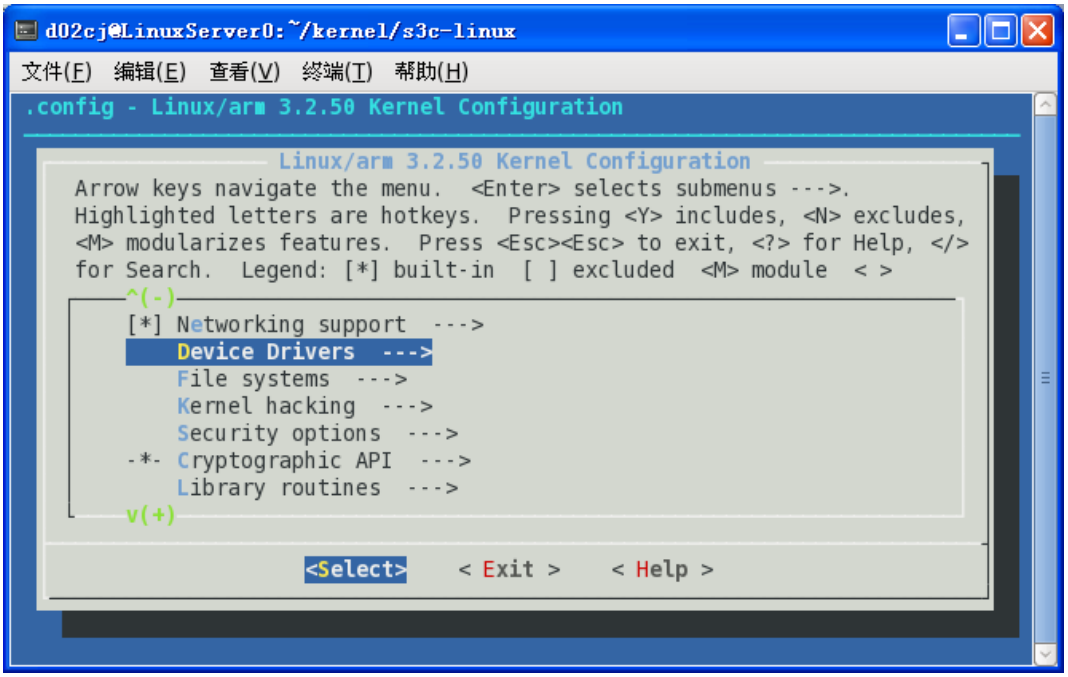


下图中选中的全选

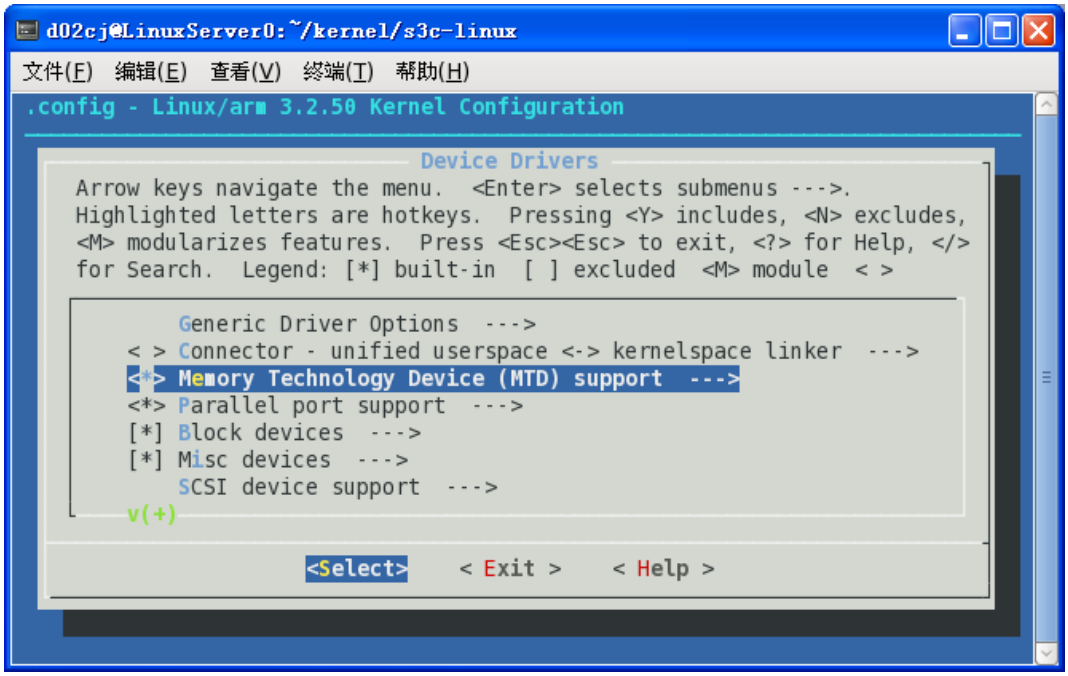


YAFFS

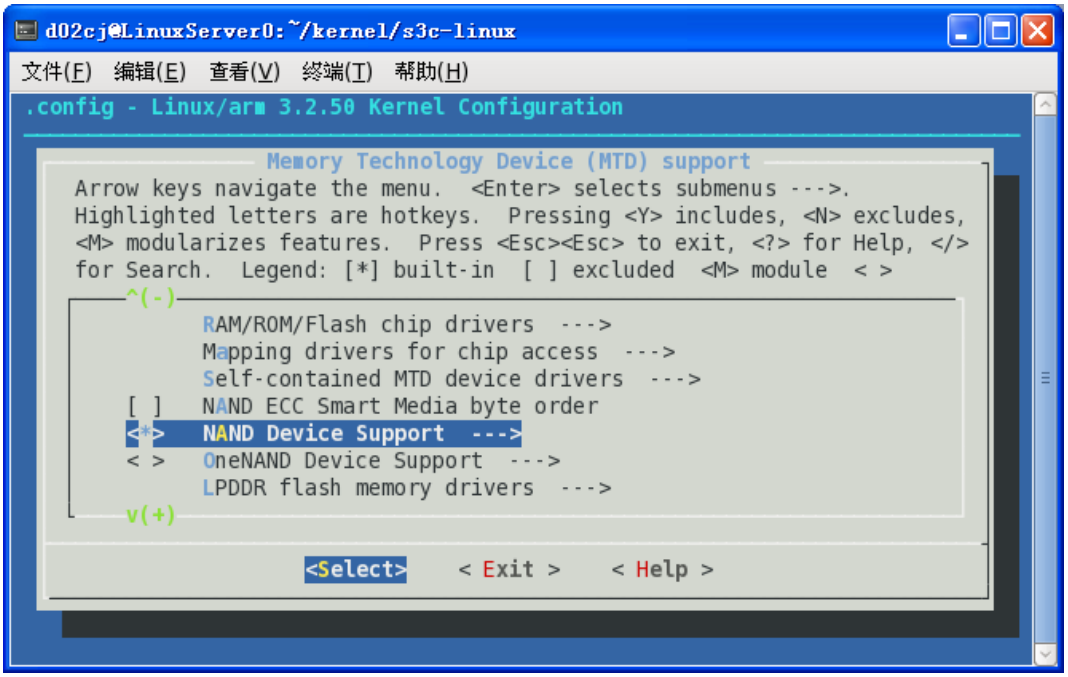
选择 Device Drivers



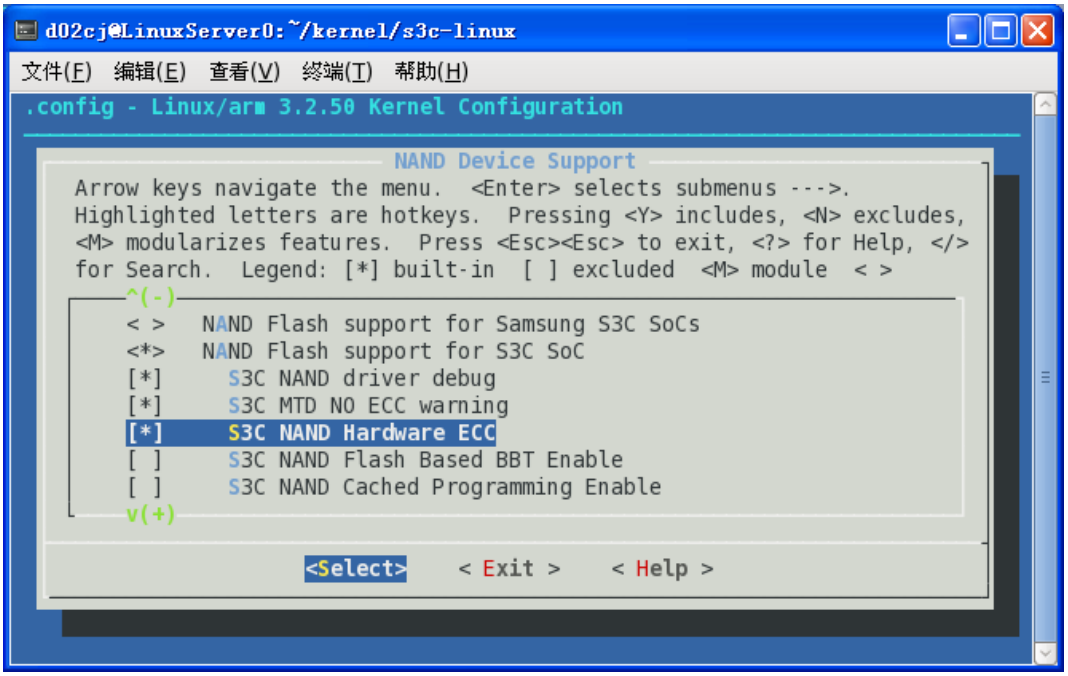
如图选中并进入 MTD：



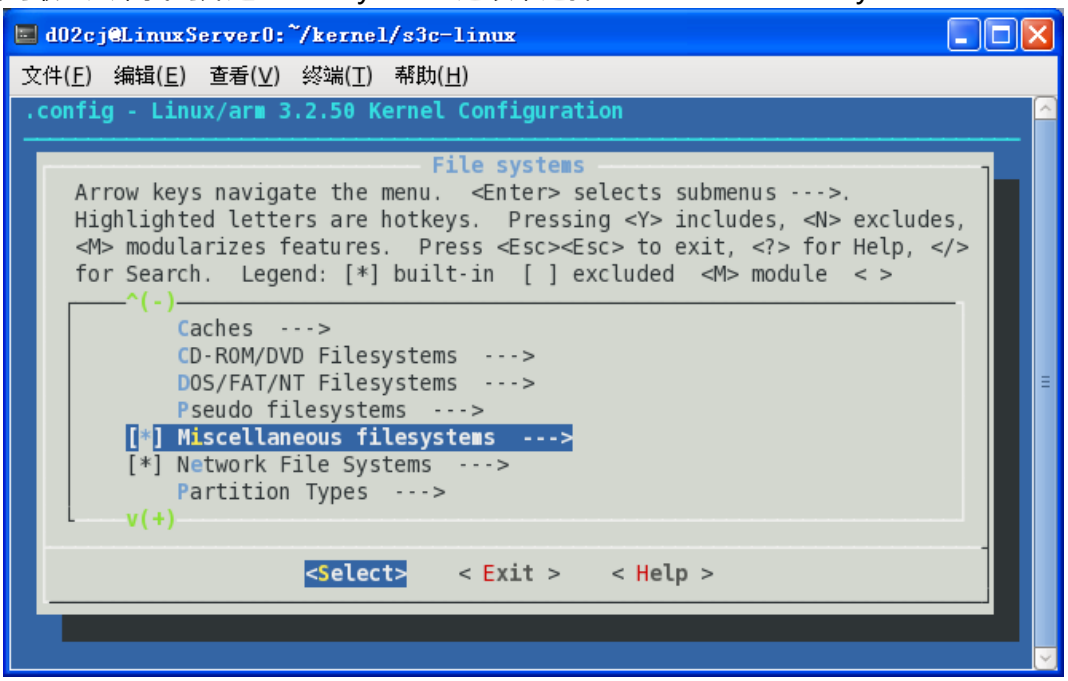
拉下来，进入 NAND Flash Device Drivers



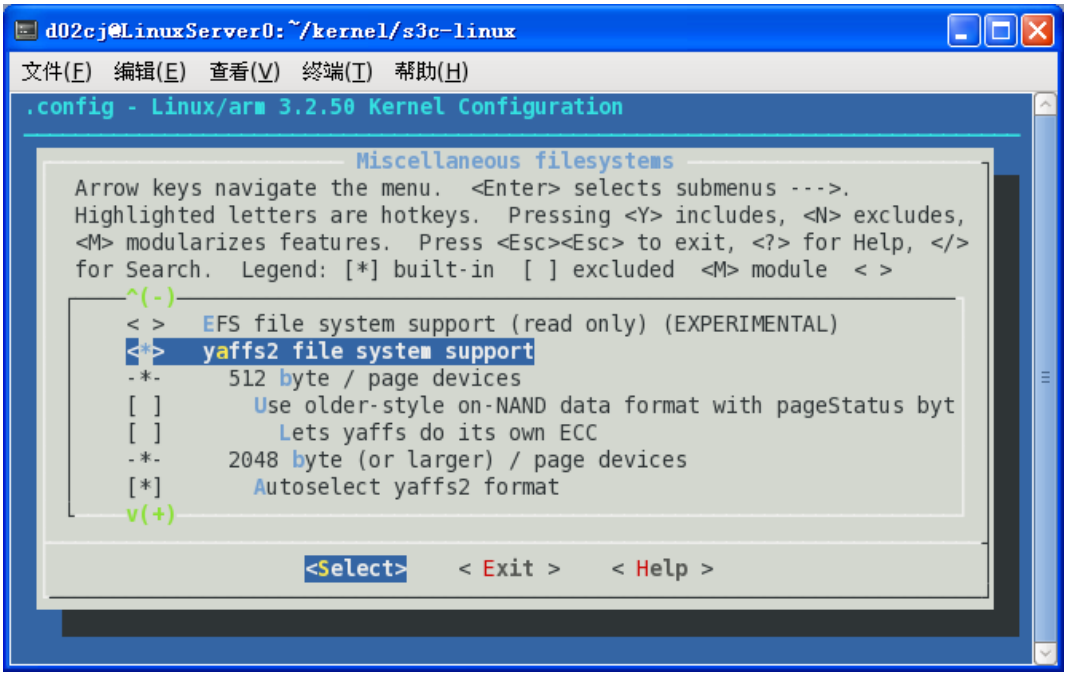
如图选择，ECC 要去 u-boot 里对应，否则会出错，这里是硬件 ECC。



返回到最上层，找到并进入 File systems 选项，选择 Miscellaneous filesystems



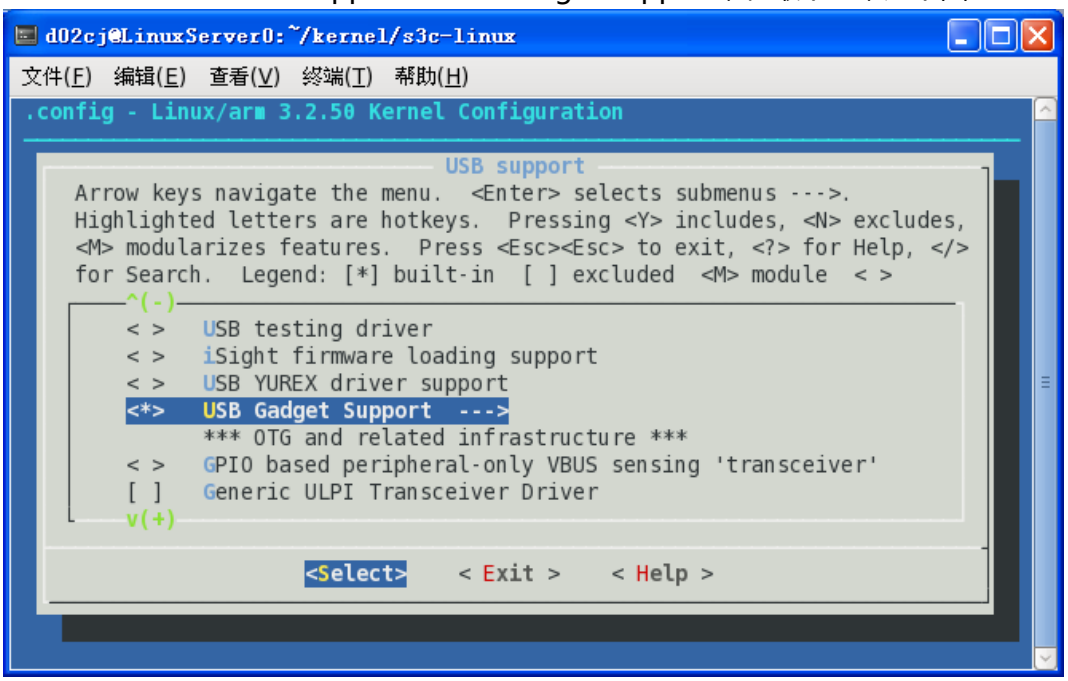
如图选中



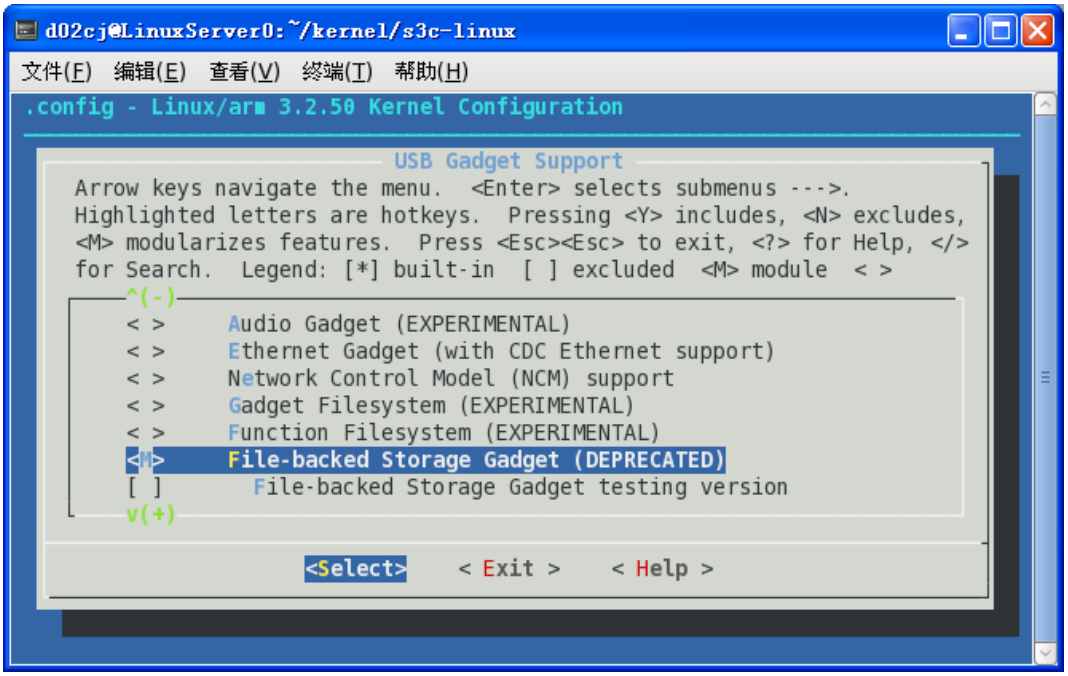
说明：yaffs2 文件系统是针对 NAND Flash 的文件系统，能自动坏块管理，官方内核是不支持的，需要自己下驱动文件打补丁到内核中。

7.8 usb-gadget 配置(将开发板当 U 盘使)

进入 Device Drivers->USB Support->USB Gadget Support，在最下边，如下图



选中 Support for USB Gadgets，再选 USB Gadget Drivers 和 File-backed Storage Gadget 为 M（多按一下空格就来了，一定要选为 M）



在 4.3 节已经介绍过如何编译和使用 usb gadget 了。

8 附录

8.1 在 VirtualBox 虚拟机里安装 linux（ ububtu 10.10 ）

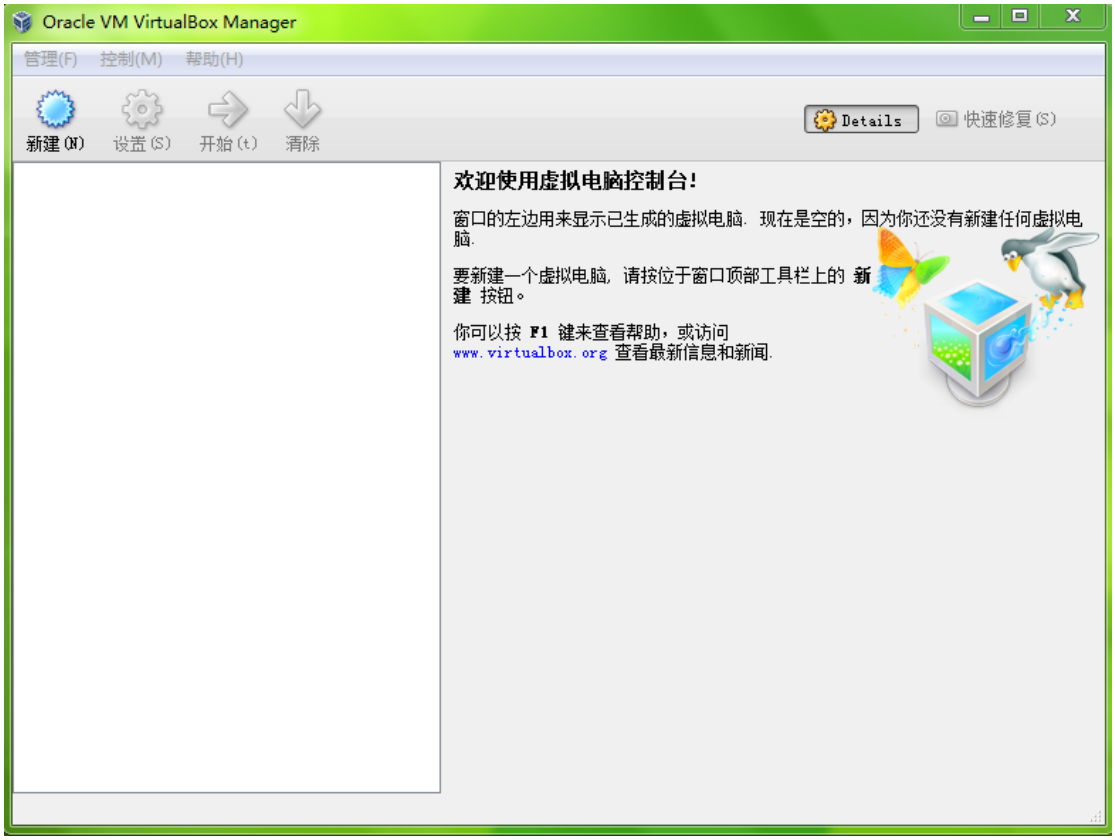
特别说明：此章节是为上一个 Helper2416 开发板编写，有些操作可能已经不能正常工作了，但是其操作方法基本实用，不同发行版本之前的安装方法略有差异。

工具：VirtualBox，官网：http://www.virtualbox.org/开源免费，而且文件体积小，不到 100M。

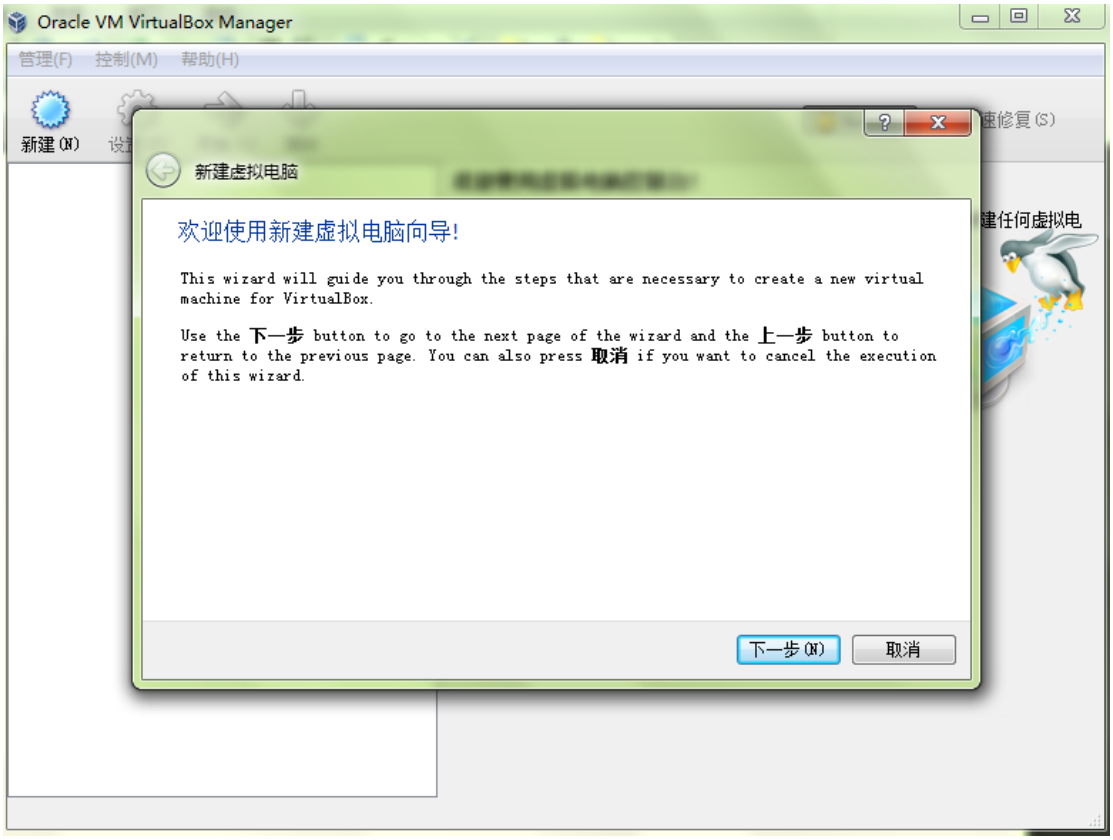
LINUX 选择 ubuntu 10.10， 下载地址：

http://download.chinaunix.net/download/0013000/12810.shtml

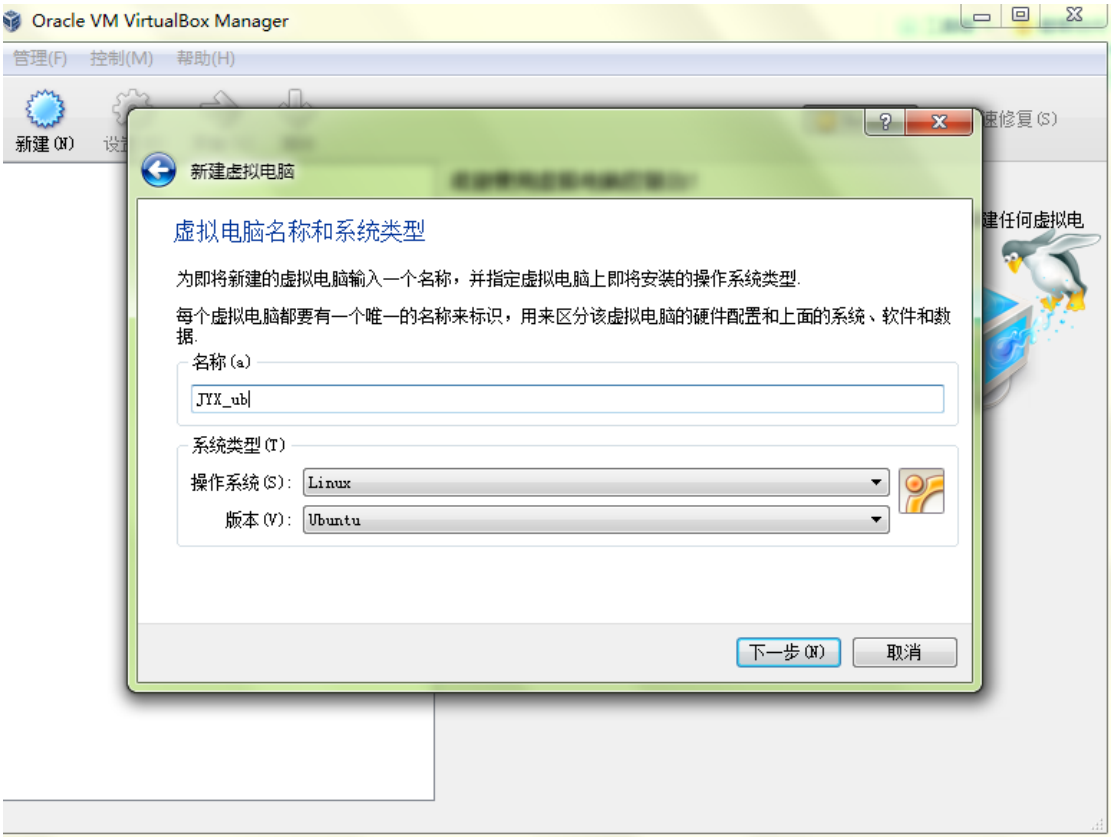
安装好 VirtualBox 后，运行效果如下：



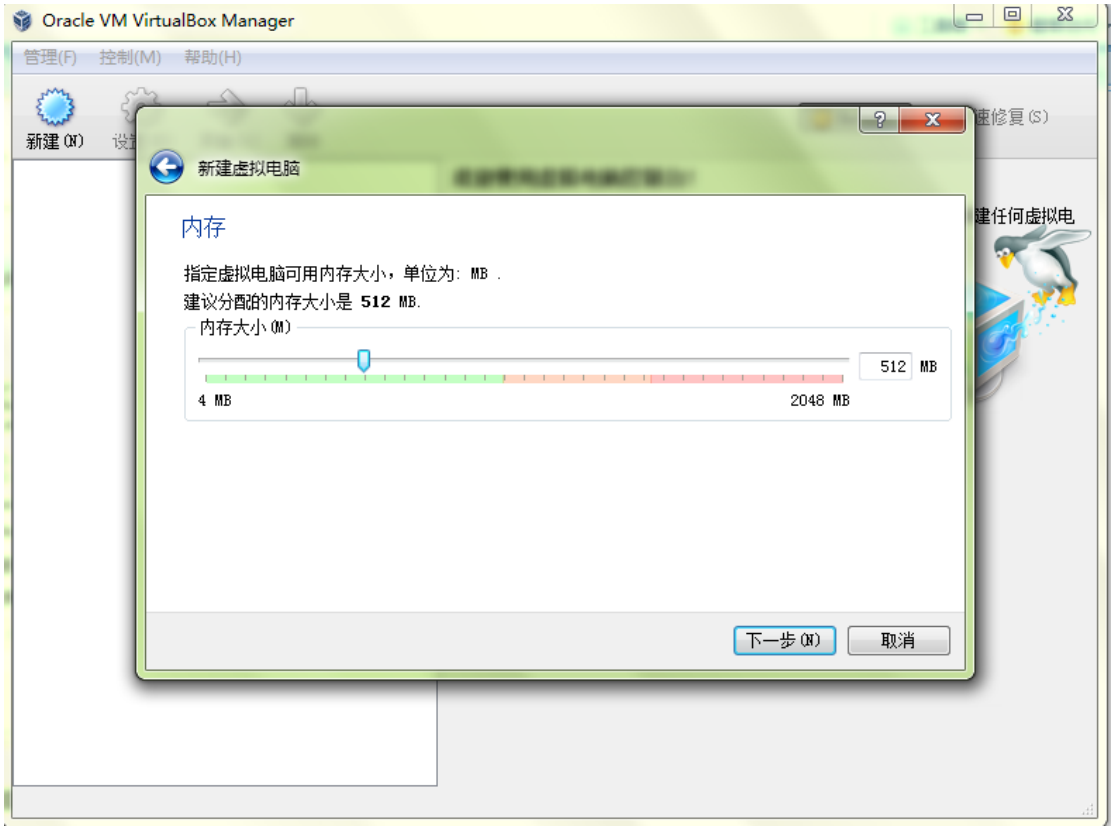
点击新建，创建虚拟机：



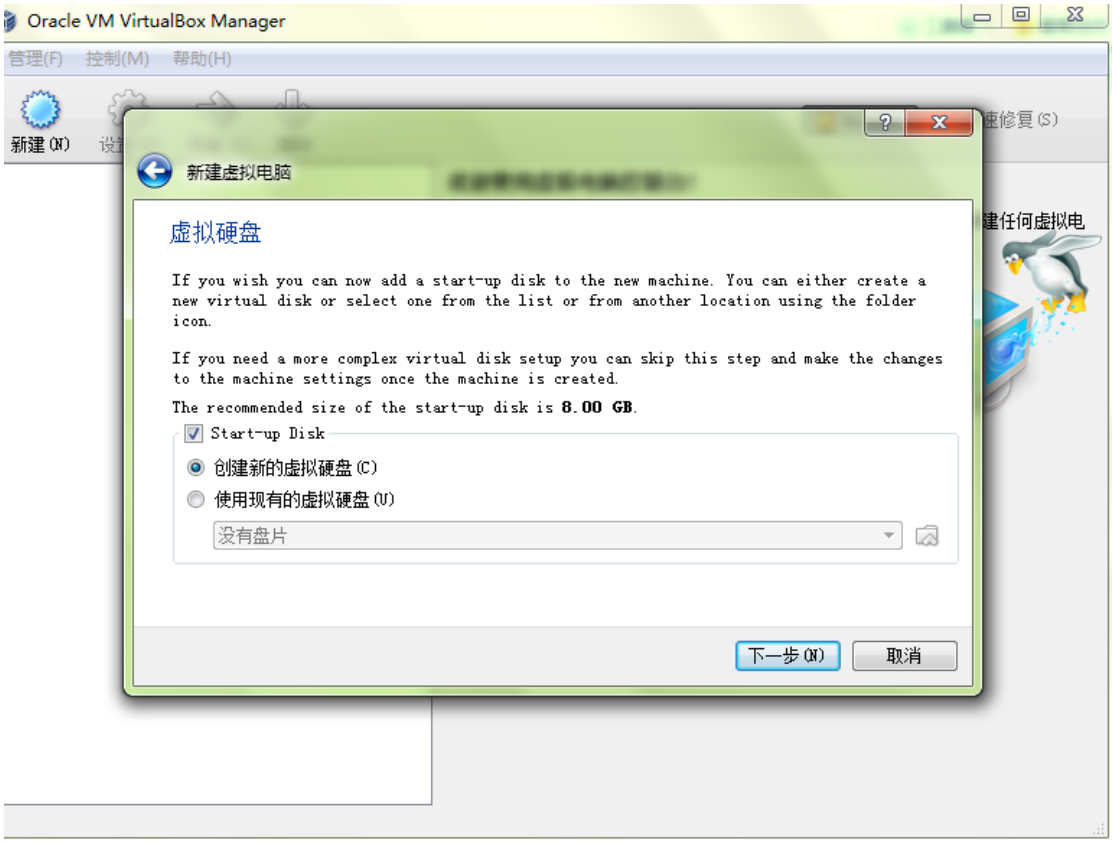
下一步：名称可以任意填，操作系统选择 linux， 版本可根据需要选择，这里选择 Ubuntu



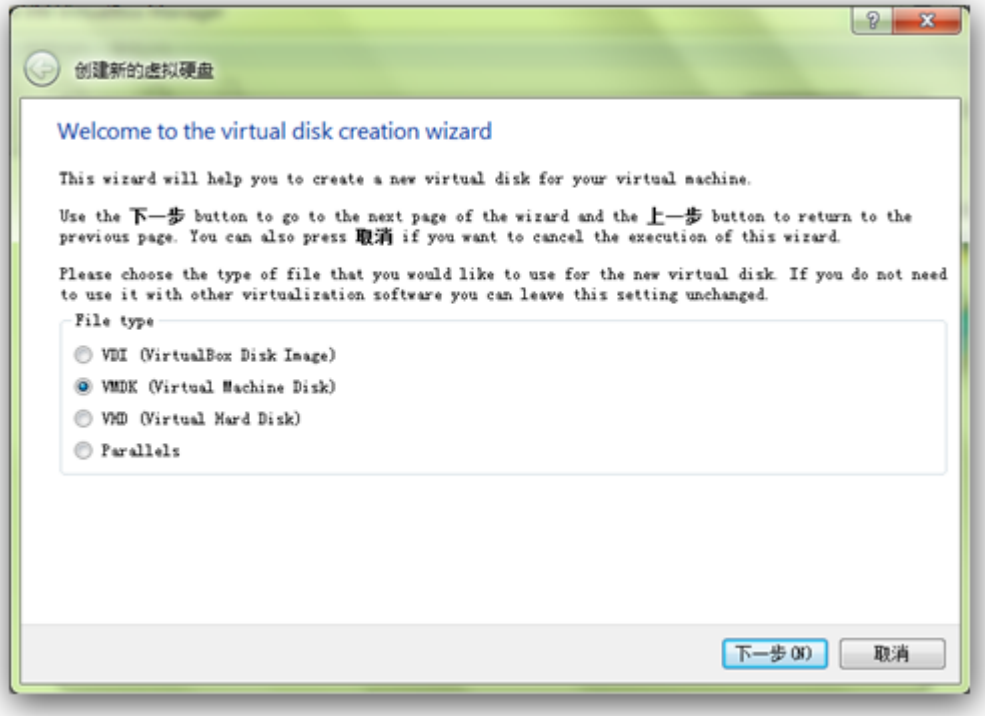
下一步：内存大小一般建议在电脑实际内存大小的 1 / 2


Oracle VM VirtualBox Manager
管理(F) 控制(M) 帮助(H)
新建(N)
新建虚拟电脑
内存
指定虚拟电脑可用内存大小，单位为: MB .
建议分配的内存大小是 512 MB.
内存大小(M)
512 MB
4 MB
2048 MB
下一步(N)
取消

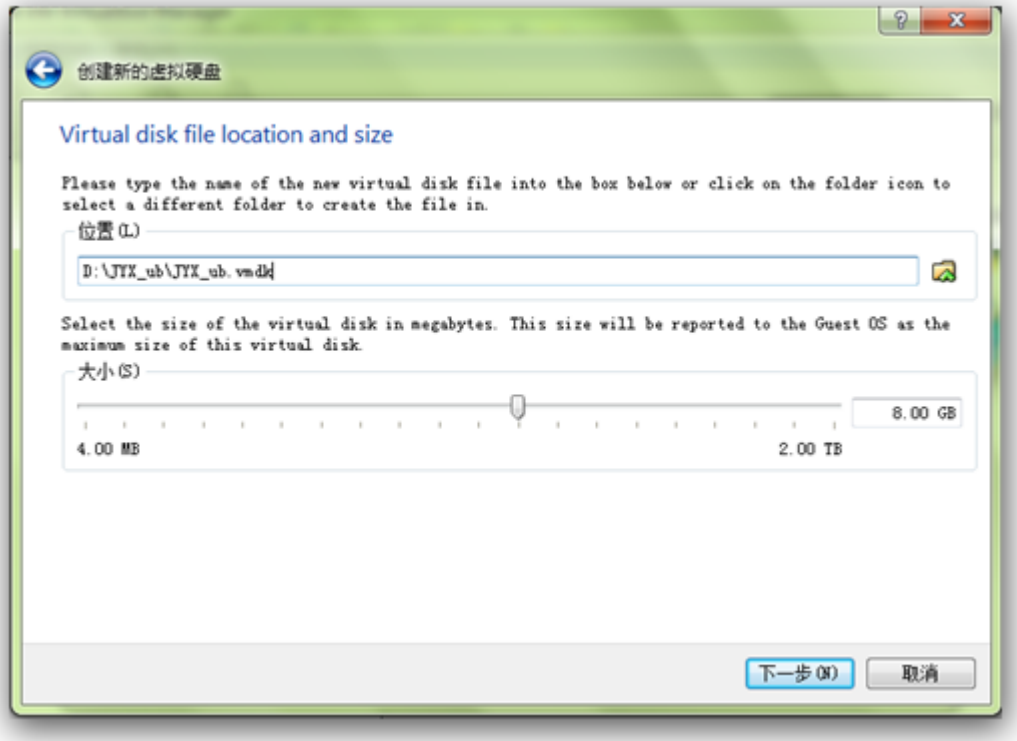
下一步：


Oracle VM VirtualBox Manager
管理(F) 控制(M) 帮助(H)
新建(N)
新建虚拟电脑
虚拟硬盘
If you wish you can now add a start-up disk to the new machine. You can either create a new virtual disk or select one from the list or from another location using the folder icon.
If you need a more complex virtual disk setup you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the start-up disk is 8.00 GB.
Start-up Disk
创建新的虚拟硬盘(C)
使用现有的虚拟硬盘(U)
没有盘片
下一步(N)
取消

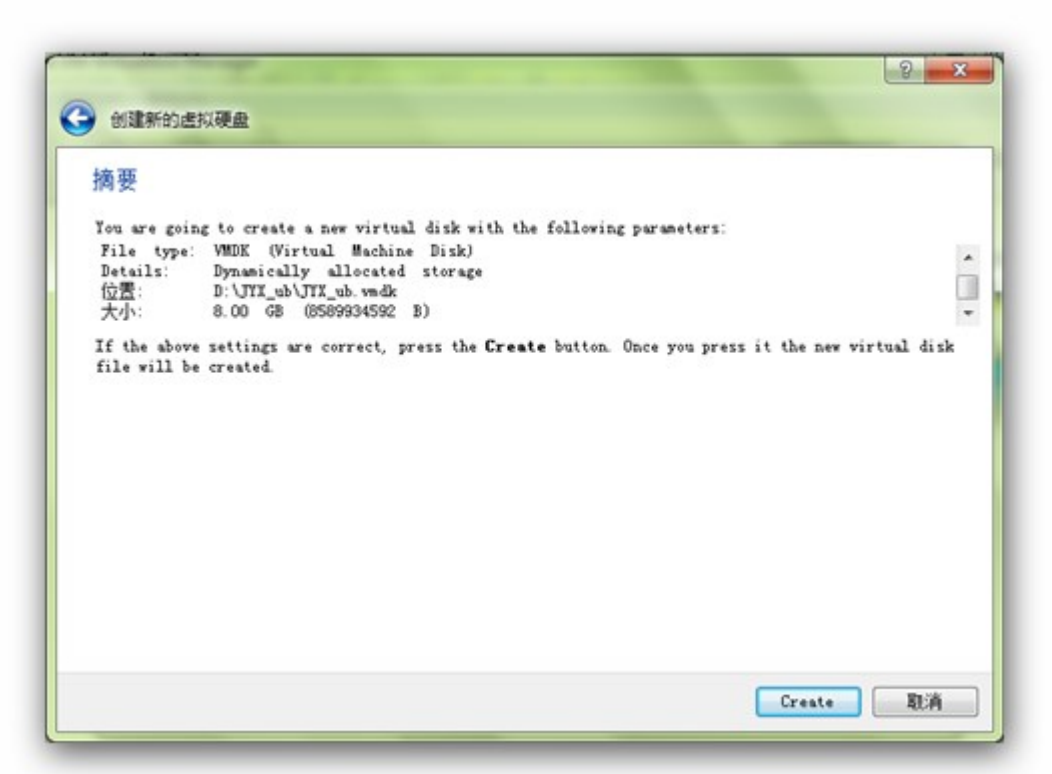
下一步：



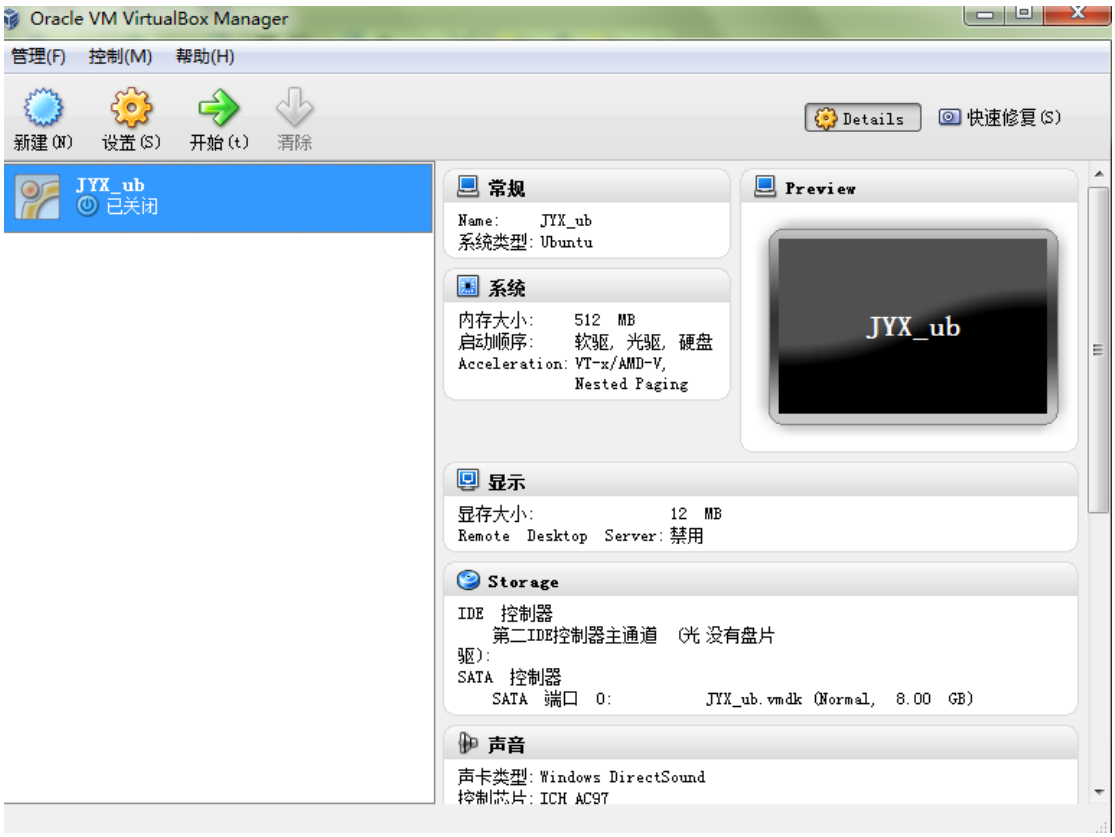
下一步，存贮位置可自己通过右边“浏览“  按钮自行选择：



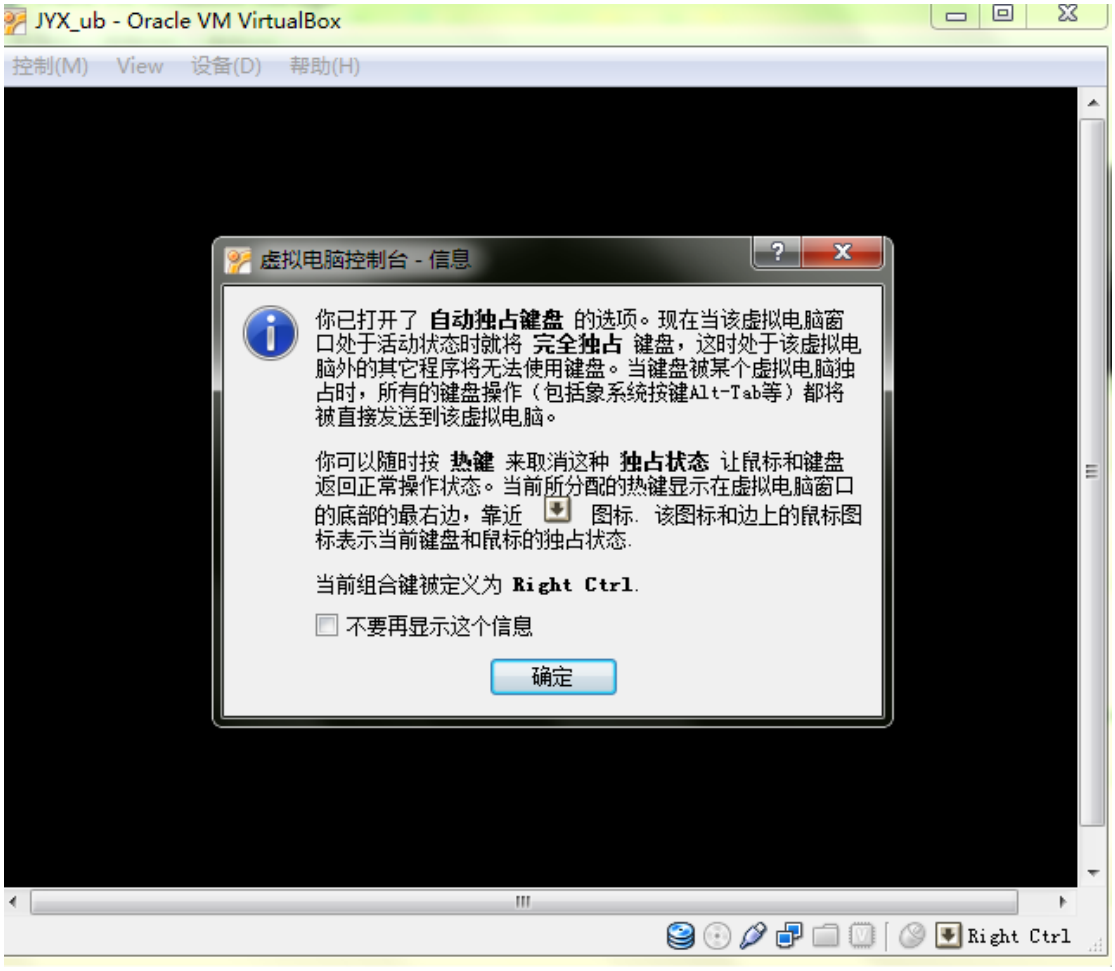
硬盘大小根据自己的需要选择，如果做会长期从事 LINUX 开发，建议硬盘选大一点，可选 32G 以上，此处仅仅是演示，选 8G

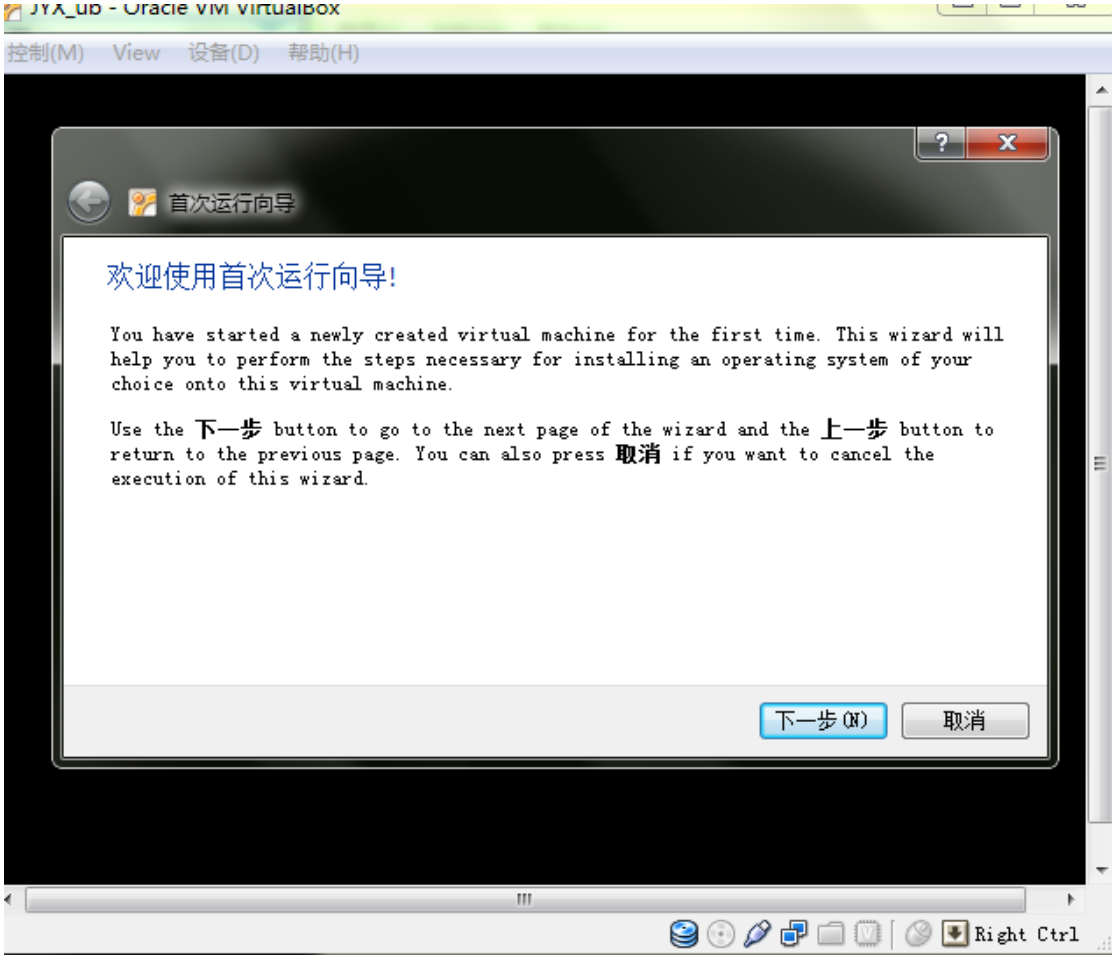


几个下一步之后得到进入如下界面：



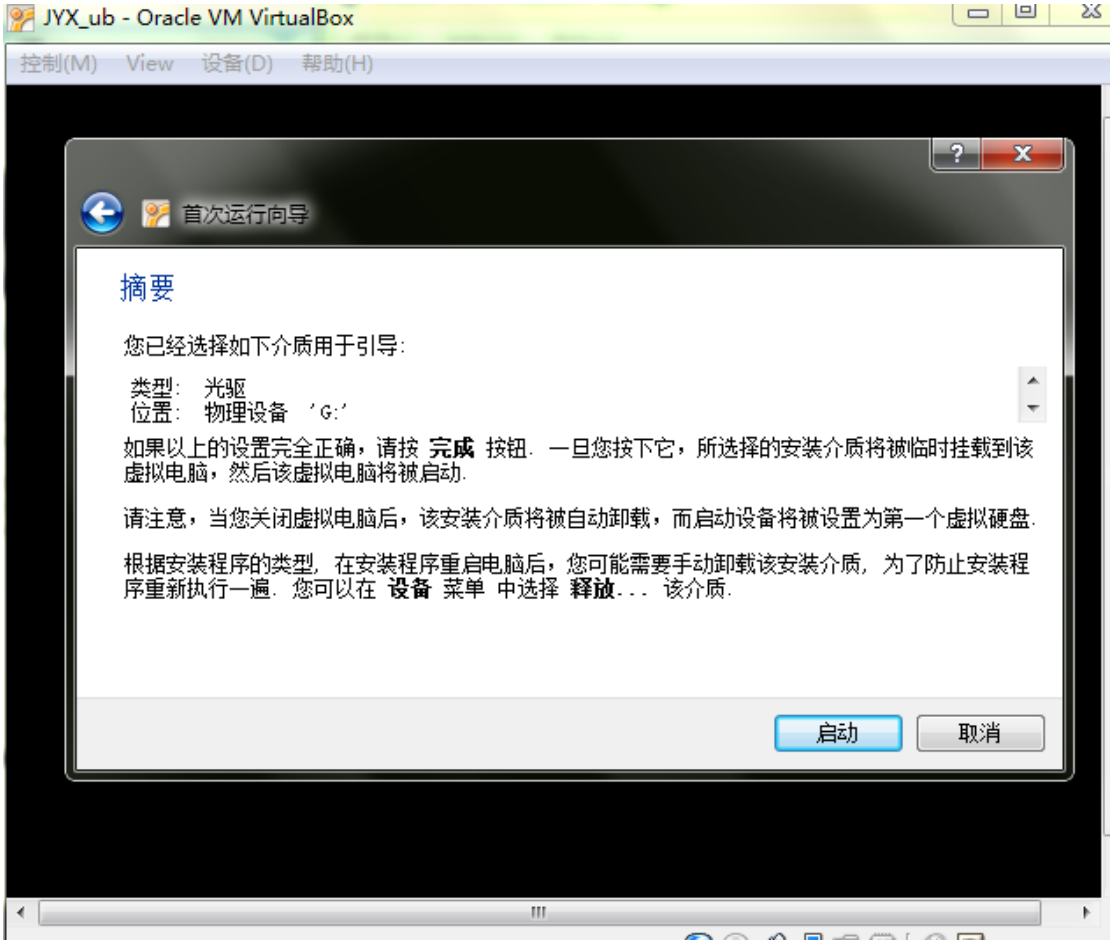
点击开始按钮：


JYX_ub - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
虚拟电脑控制台 - 信息
你已打开了 自动独占键盘 的选项。现在当该虚拟电脑窗口处于活动状态时就将 完全独占 键盘，这时处于该虚拟电脑外的其它程序将无法使用键盘。当键盘被某个虚拟电脑独占时，所有的键盘操作（包括象系统按键Alt-Tab等）都将被直接发送到该虚拟电脑。
你可以随时按 热键 来取消这种 独占状态 让鼠标和键盘返回正常操作状态。当前所分配的热键显示在虚拟电脑窗口的底部的最右边，靠近 图标. 该图标和边上的鼠标图标表示当前键盘和鼠标的独占状态.
当前组合键被定义为 Right Ctrl.
不要再显示这个信息
确定
Right Ctrl



下一步：选择 iso 安装包，使用最右的浏览按钮进行定位

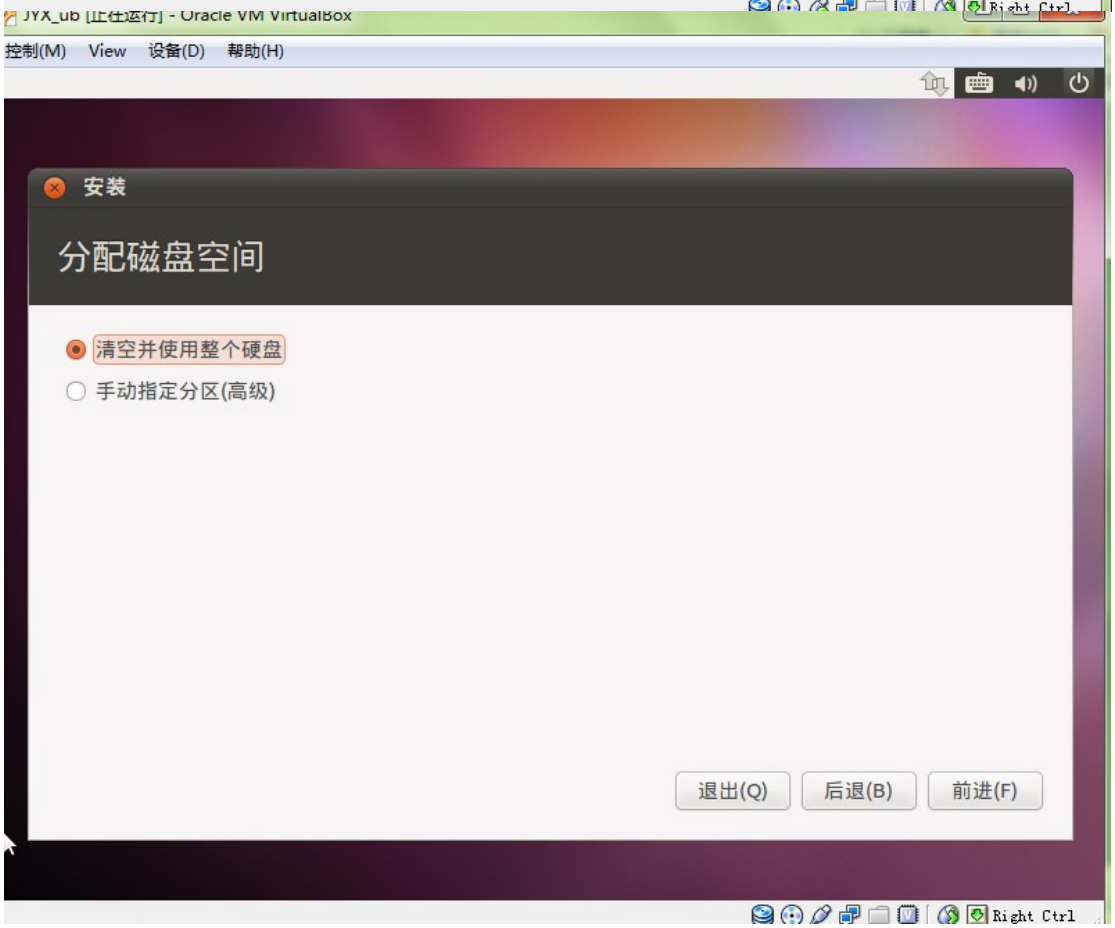

JYX_ub - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
首次运行向导
选择安装介质
选择含有您所要安装的操作系统的安装程序的介质。该介质必须是可引导的，否则安装程序无法启动.
介质位置
ubuntu-10.10-desktop-i386.iso (693.16 MB)
下一步(N)
取消
Right Ctrl


JYX_ub - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
首次运行向导
摘要
您已经选择如下介质用于引导:
类型: 光驱
位置: 物理设备 'G:'
如果以上的设置完全正确，请按 完成 按钮. 一旦您按下它，所选择的安装介质将被临时挂载到该虚拟电脑，然后该虚拟电脑将被启动.
请注意，当您关闭虚拟电脑后，该安装介质将被自动卸载，而启动设备将被设置为第一个虚拟硬盘.
根据安装程序的类型，在安装程序重启电脑后，您可能需要手动卸载该安装介质，为了防止安装程序重新执行一遍. 您可以在 设备 菜单 中选择 释放... 该介质.
启动
取消

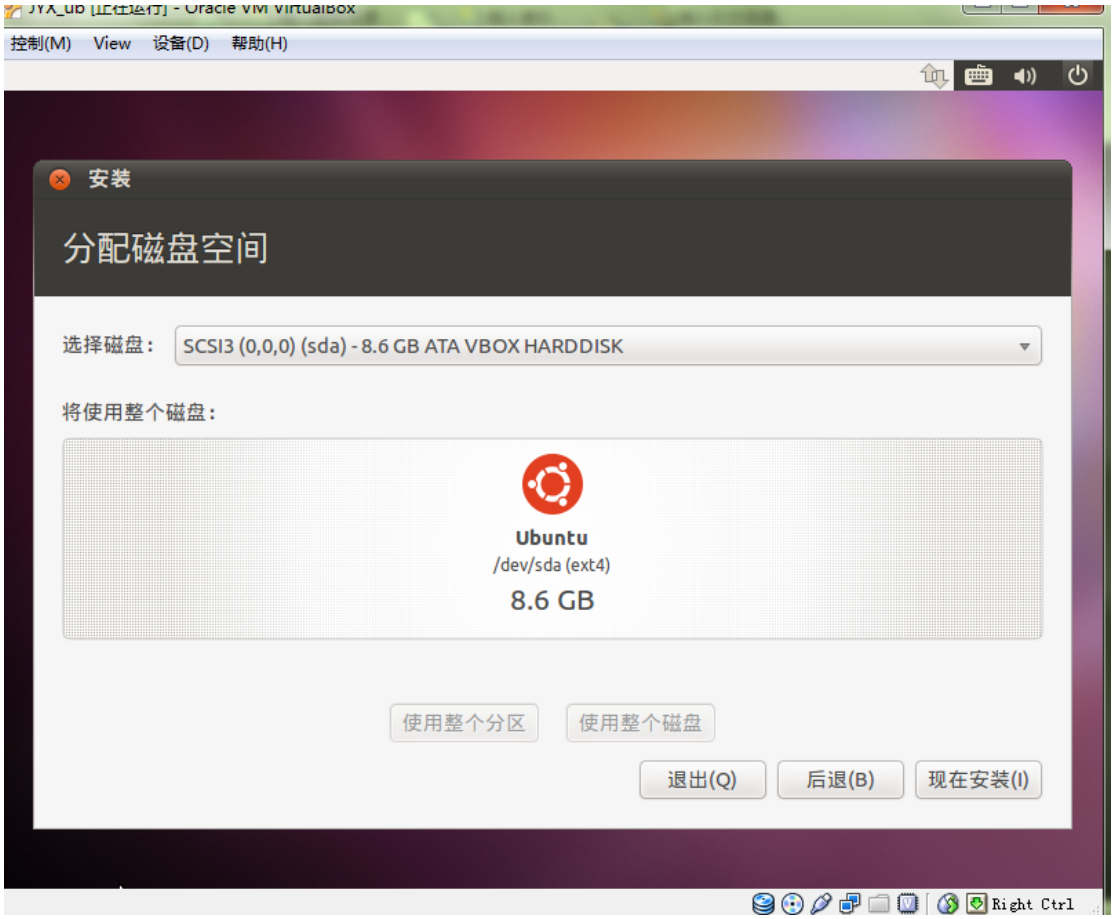

JVX_ub [正在运行] - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
安装
欢迎
ქართული
עברית
عربي
አማርኛ
नेपाली
मराठी
हिन्दी
বাংলা
ਪੰਜਾਬੀ
ગુજરાતી
தமிழ்
മലയാളം
ภาษาไทย
한국어
中文(简体)
中文(繁體)
试用 Ubuntu
安装 Ubuntu
您可以直接从此 CD 尝试 Ubuntu，而不用对您的电脑作任何更改。
如果您已经准备完毕，您可以与现有系统并存（或者替代）方式将Ubuntu 安装到您的电脑上。此过程无需耗时太久。
您可能希望阅读 发行注记。

点击安装 Ubuntu


JYX_ub [正在运行] - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
安装
准备安装 Ubuntu
要获得最佳的体验，请确定这台计算机：
有至少 2.6 GB 可用的磁盘空间
已经插入电源
已经连接到互联网
安装中下载更新
Ubuntu 使用第三方的软件处理 Flash、MP3 等媒体文件，驱动一些无线设备。这其中有一些软件是闭源的。这些软件受软件文档附带的许可协议限制。
安装这个第三方软件
Fluendo MP3 插件包含 Fraunhofer IIS 和 Thomson 授权的 MPEG Layer-3 音频解码技术。
退出(Q)
后退(B)
前进(F)


控制(M) View 设备(D) 帮助(H)
安装
分配磁盘空间
清空并使用整个硬盘
手动指定分区(高级)
退出(Q)
后退(B)
前进(F)
Right Ctrl

如果要自己分区，可选择手动指定分区。




JYX_ub [正在运行] - Oracle VM VirtualBox
控制(M) View 设备(D) 帮助(H)
安装
您在什么地方？
13:33
Geonames.org
Chongqing [在此输入以更改]
后退(B)
前进(F)
就绪，在您...
Right Ctrl

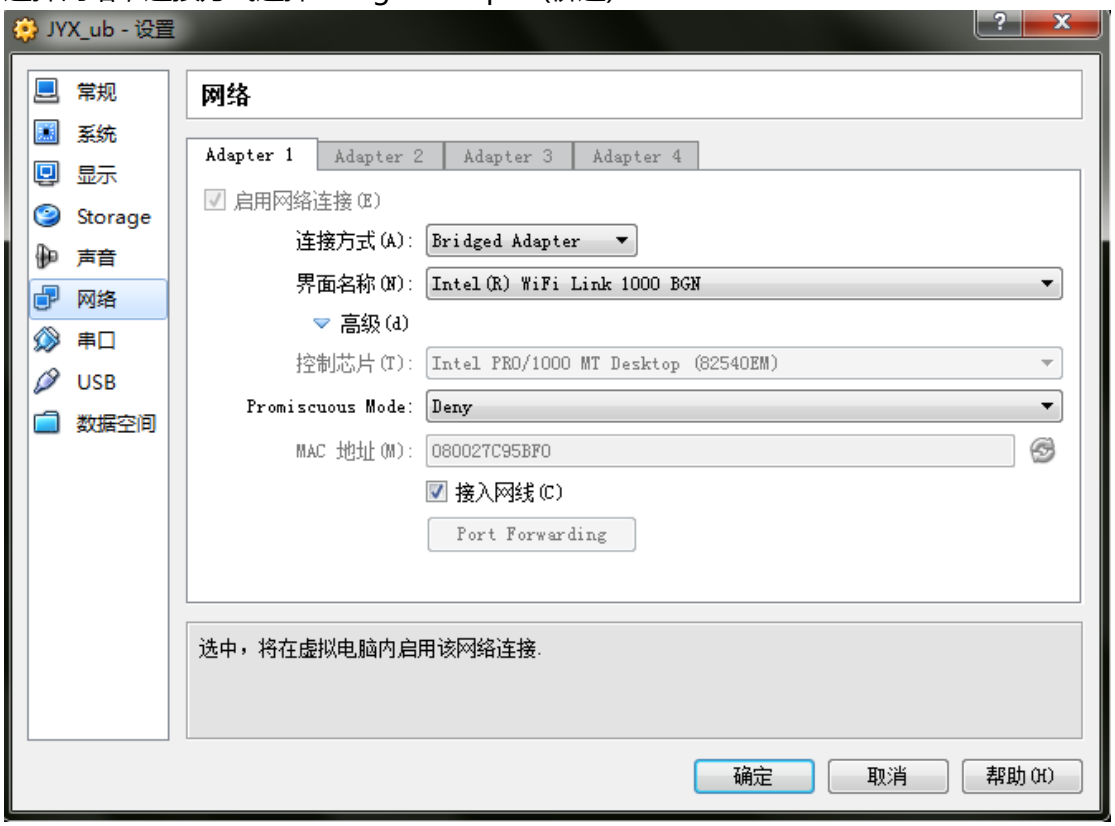
键盘布局选择默认

这里可自行指定填入，前进，等待安装。

8.1.1 配置虚拟机网络连接

控制->设置，如下图：

选择网络，连接方式选择 Bridged Adapter(桥连)

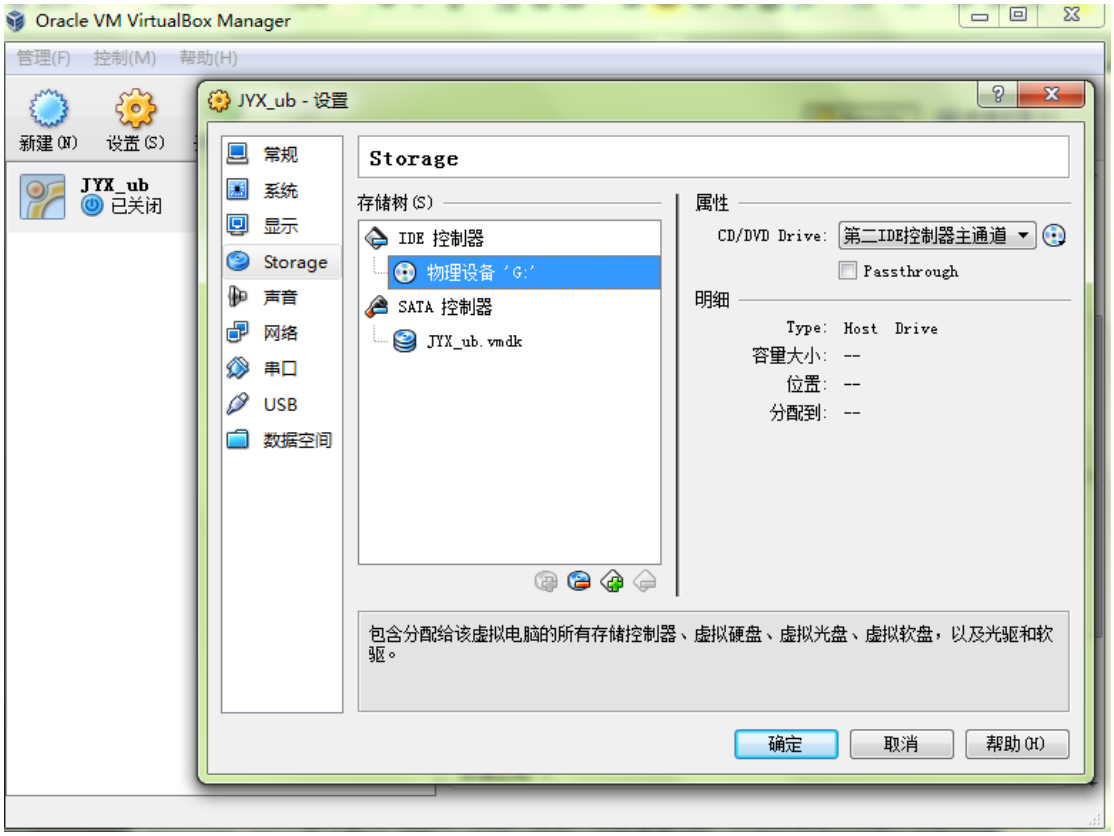


特别说明：如果你的电脑与开发板是通过网线直接相连的话，请一定设置上图的界面名称一栏为与开发板相连的网卡，否则开发板可能无法访问虚拟机。

如果连接方式使用 NAT，虚拟机也可访问网络，但是 ip 会与主机不在一个网段，虚拟机的网络相对独立，这样开发板也不能访问虚拟机。

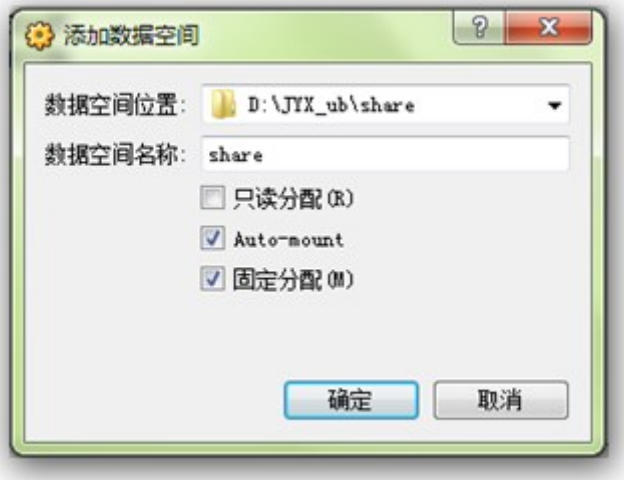
8.1.2 光驱设置

点击设置，最右边属性处点击光盘图标，选择物理设备，并点击确定，这样重启后不会再进入安装模式。



8.1.3 主机与虚拟机间共享文件

首先在 virtualbox 选择 设备->安装增强功能，然后根据提示安装；安装完成后选择 设备->分配数据空间->固定分配



结束后，必须重启，因为选择了 Auto-mount，所以就不用手动去 mount 了，我们必须知道它被自动挂载到哪了，在终端中输入 mount | grep vboxsf 会得到挂载的位置

```
jyx@jyx-VirtualBox:~$ mount | grep vboxsf
```

```
share on /media/sf_share type vboxsf (gid=1001，rw)
```

此例是位于/media/sf_share 下，这样，虚拟机就可以通过/media/sf_share 与主机实现共享了。

注意：此处的共享，在 LINUX 下就是一个目录，即：/media/sf_share，但是，必须是 root 用户才能访问。

另外，您也可以使用 samba 与 windows 互访，具体操作可以在网上查询，此处不再赘述。

在 linux 下做软件开发，大多数情况都是在虚拟终端下操作，在 ubuntu 10.10 下，点击 应用程序->附件->终端，即进入虚拟终端窗口：

注：我们后边介绍的执行命令默认都是在虚拟终端下进行。

8.1.4 安装开发环境

安装 PC 端 gcc 等编译工具，如下用到 sudo 命令

```
$ sudo apt-get install build-essential uuid-dev libxmu-dev libxmu6 zlib1g-dev libz-dev libncurses5-dev gperf -y
```

安装 bison 语法解析器、flex 词法解析器

```
$ sudo apt-get install bison flex -y
```

安装 automake libtool，自动生成 Makefile 会用到

```
$ sudo apt-get install automake libtool -y
```

安装一些 x11 开发库

```
$ sudo apt-get install libx11-dev libxext-dev -y
```

下面这个包与上面 x11 开发库在编译 qtopia 时要用到

```
$ sudo apt-get install libqt3-mt-dev -y
```

下面这个包是给编译器用的，正常设置后，能够成倍提高编译速度

```
$ sudo apt-get install ccache -y
```

8.1.5 安装交叉编译器

请一定使用我们提供的编译器，我们不保证我们的源码在其它编译器下工作正常。

软件包目录里有本公司制作的，针对 ARM926EJ-S 优化的交叉编译器，在 compiler/arm926ej-eabi-4.4.6.tar，解压缩到任意目录都可以运行。我们现在安装到系统目录，这样比较好实现多个用户共享，例如：/opt/toolchains/目录，假设光盘加载到：/media/20120809_HELPER2416 先拷贝到当前目录，然后解压：

```
$ cp /media/20120809_HELPER2416/Helper2416/compiler/arm926ej-eabi-4.4.6.tar .
$ sudo mkdir -p /opt/toolchains
$ sudo tar -xvf   arm926ej-eabi-4.4.6.tar -C   /opt/toolchains/
```

设置环境变量：

```
$ gedit ~/.bashrc
```

打开.bashrc 并在后面加上一行(注意：下边内容是一行，复制时一定不能有换行)

```
export PATH=/opt/toolchains/arm-jyxtec-linux-gnueabi/ccache:/opt/toolchains/arm-jyxtec-linux-gnueabi/bin:$PATH
```

保存并关闭文件，执行

```
$ source ~/.bashrc
```

则立即生效，此后重新登录就自动运行了。

运行以下命令 arm-linux-gcc -v，如果提示如下信息，则说明安装成功。

```
 $  arm-linux-gcc -v
Target: arm-jyxtec-linux-gnueabi
Configured with: /home/d02cj/arm/crosstool_build/.build/src/gcc-4.4.6/configure --build=i686-build_pc-linux-gnu --host=i686-build_pc-linux-gnu --target=arm-jyxtec-linux-gnueabi --prefix=/home/d02cj/x-tools/arm-jyxtec-linux-gnueabi --with-sysroot=/home/d02cj/x-tools/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot --enable-languages=c,c++ --disable-multilib --with-arch=armv5te --with-cpu=arm926ej-s --with-tune=arm926ej-s --with-float=soft --with-pkgversion='crosstool-NG 1.13.3' --disable-sjlj-exceptions --enable-__cxa_atexit --disable-libmudflap --disable-libgomp --disable-libssp --with-gmp=/home/d02cj/arm/crosstool_build/.build/arm-jyxtec-linux-gnueabi/build/static --with-mpfr=/home/d02cj/arm/crosstool_build/.build/arm-jyxtec-linux-gnueabi/build/static --with-ppl=/home/d02cj/arm/crosstool_build/.build/arm-jyxtec-linux-gnueabi/build/static --with-cloog=/home/d02cj/arm/crosstool_build/.build/arm-jyxtec-linux-gnueabi/build/static --enable-threads=posix --enable-target-optspace --with-local-prefix=/home/d02cj/x-tools/arm-jyxtec-linux-gnueabi/arm-jyxtec-linux-gnueabi/sysroot --disable-nls --enable-c99 --enable-long-long
Thread model: posix
gcc version 4.4.6 (crosstool-NG 1.13.3)
```

8.2 注意事项

如果您想把液晶屏用螺丝固定到主板上，请千万不要用螺柱垫高主板，因为设计时为了满足良好的阻抗匹配，主板做得比较薄只有 1mm 的厚度，因此垫高主板容易引起主板变形。

重新编译 qtopia，并替换根文件系统时，必须拷贝 zlib 的库文件与字体文件（参见 4.6.6 交叉编译 qtopia 一节）。

8.3 编译小技巧

在编译大型工程的时候，通常是用 make 命令，而对于编译主机是多核 CPU 的，请加上一个-j 参数，如下：

```
$ make -j4
```

表示是用 4 线程编译，适用于 4 线程 CPU，编译速度会成倍提高。

如果觉得 make 的时候提供信息太多，很多无用信息淹没了有用信息，则加上一个-s 参数：

```
$ make -s
```

这样，make 程序只会提供警告以及错误信息。

在编译完成后，得到的最终执行文件里，通常还会有一些杂七杂八的信息，使执行文件比较大，可以用 GCC 下的压缩工具进行压缩，通常压缩之后文件会小一半以上，而且加载速度更快了。在 PC 机上，压缩工具是 strip，在 arm 平台上通常是 arm-linux-strip，执行方法如下：

```
$ arm-linux-strip serial_test
```

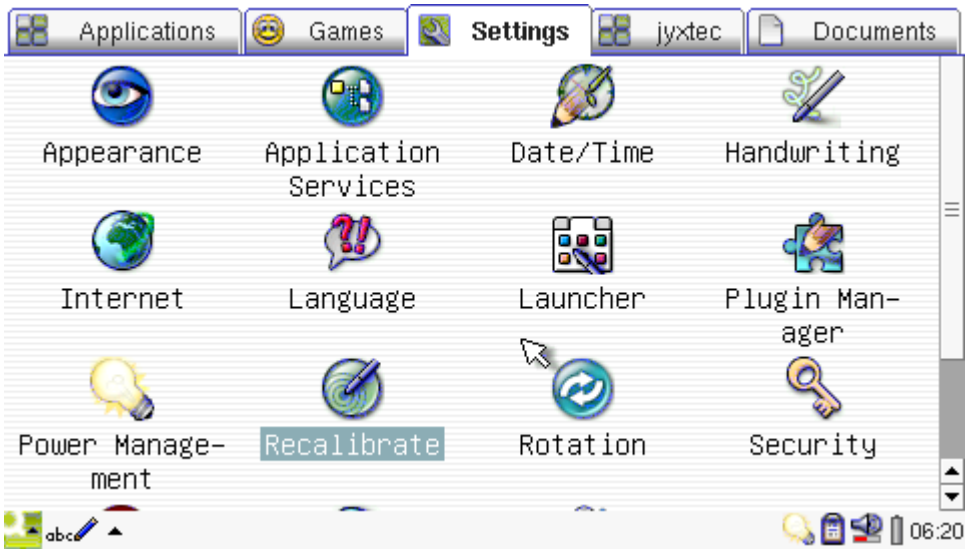
serial_test 指的是被压缩文件，可以是执行文件，也可以是动态链接库。

还有一个关于编译很重要的东西，就是 ccache，前面有介绍安装方法，使用 ccache，也可以成倍提高编译速度。对于我们提供的 ARM 编译器已经做了支持，只要按照 3.2 节的方法安装就已经支持，至于 ccache 的原理，用户可自行到网上了解。

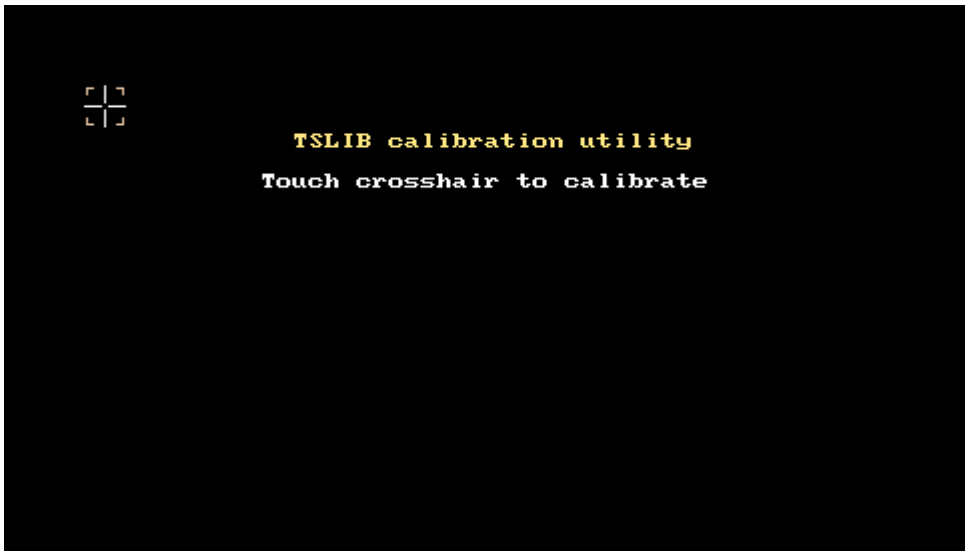
8.4 触摸屏校准

有时候触摸屏校准时不小心按错了，或者用的时间久了电阻屏发生飘移，需要重新校准，如下是两种进入校准的方法。

1．在 USB Host 接口插入 USB 鼠标，重新启动开发板，之后再点击屏幕上的 Settings，再点击 Recalibrate，再点 OK 进入校准程序，如下图：



用普通笔状物，或者指甲依次点击“十“字即可。如下图：



2．连接开发板调试串口从超级终端操作目标板文件，删除/etc/pointercal，重新启动开发板会再次进入校准程序，按上一节的方法校准。如下：

```
$ rm /etc/pointercal
$ reboot
```

8.5 关于触摸屏与鼠标共存的问题

在根文件系统的/bin 目录有一个 qtopia 的执行脚本，里边有一段内容是：

```
export QWS_MOUSE_PROTO="TPanel:$INPUT_PATH/event1
USB:$INPUT_PATH/mouse1"
```

在根目录下也有一个脚 switch_to_qt4，里边同样有一段内容是：

```
export QWS_MOUSE_PROTO="Tslib:$INPUT_PATH/event1
MouseMan:$INPUT_PATH/mouse1"
```

两个都是描述触摸屏设备和鼠标设备的，qtopia 是为 qtopia2.2 服务的，switch_to_qt4 是为 QT4 服务的，使用的时候要注意，qtopia2.2 和 QT4 是不一样的。

在我们的内核里已经固定内部键盘对应的设备地址是/dev/input/event0，内部触摸屏对应的设备地址是/dev/input/event1，因此我们设置 QT 时，触摸屏设备可以固定为/dev/input/event1，就像上边的脚本写的一样。但是如果是插入 USB 鼠标，其对应的设备地址则是可变的，第一次插入的时候地址应该是/dev/input/event2，拔出再插的时候设备地址变为/dev/input/event3，再插就会依次累加。这就决定了我们的脚本不支持拔出再插鼠标的情况。

本来，对于鼠标设备有一个通用设备地址是/dev/input/mice，这个可以响应任何鼠标操作，不管插入多少次，都会在该设备上有相应的事件，为什么我们不用这个设备呢？因为触摸屏也被规到了鼠标设备一类，如果我们使用了 mice 设备，当 QT 处理触摸屏事件的时候，同时也会处理鼠标设备事件，这样就引起混乱了。最终造成触摸屏会抖动。

友善之臂的根文件系统用的是 mice 这个共享设备，所以他们的根文件系统运行 QT 应该是会抖动的，在 qtopia 里的 Notes 程序里，使用手写识别输入的时候应该可以看出来。

当然，有人说，触摸屏也可以直接用鼠标设备，但是如果 QT 直接使用 mice 设备的话，就不能使用 tslib 的校准功能了。

8.6 一个 LINUX 应用程序例子--抓屏软件源码

本手册所有目标板屏幕截图都是由该抓屏软件得到，现提供源码如下：

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
static unsigned char sg_BHeader[] = {
    0x42， 0x4D， 0x36， 0x58， 0x02， 0x00， 0x00， 0x00， 0x00， 0x00，
0x36， 0x00， 0x00， 0x00， 0x28， 0x00，
    0x00， 0x00， 0x40， 0x01， 0x00， 0x00， 0xF0， 0x00， 0x00， 0x00， 0x01，
0x00， 0x10， 0x00， 0x00， 0x00，
    0x00， 0x00， 0x00， 0x00， 0x00， 0x00， 0x00， 0x00， 0x00， 0x00， 0x00，
0x00， 0x00， 0x00， 0x00， 0x00，
    0x00， 0x00， 0x00， 0x00， 0x00， 0x00
};
#define RGB565TO1555(rgb) ((unsigned short)((unsigned short)(rgb & 0x001f) | ((unsigned
short)(rgb & 0xffe0) >> 1)))

void SaveBMPFile(unsigned char *raw， char *filename， int m_Width， int m_Height)
{
    unsigned short *p = (unsigned short *)raw;
```

```
    typedef unsigned int UINT;
    typedef unsigned char UCHAR;
    UINT i， j;
    int bmp = open(filename， O_WRONLY | O_CREAT， S_IRWXU | S_IRGRP |
S_IROTH);
    if(bmp < 0)
        return;
    sg_BHeader[0x02] = (UCHAR)(m_Width * m_Height * 2 + 0x36) & 0xff;
    sg_BHeader[0x03] = (UCHAR)((m_Width * m_Height * 2 + 0x36) >> 8) & 0xff;
    sg_BHeader[0x04] = (UCHAR)((m_Width * m_Height * 2 + 0x36) >> 16) & 0xff;
    sg_BHeader[0x05] = (UCHAR)((m_Width * m_Height * 2 + 0x36) >> 24) & 0xff;
    sg_BHeader[0x12] = (UCHAR)m_Width & 0xff;
    sg_BHeader[0x13] = (UCHAR)(m_Width >> 8) & 0xff;
    sg_BHeader[0x14] = (UCHAR)(m_Width >> 16) & 0xff;
    sg_BHeader[0x15] = (UCHAR)(m_Width >> 24) & 0xff;
    sg_BHeader[0x16] = (UCHAR)m_Height & 0xff;
    sg_BHeader[0x17] = (UCHAR)(m_Height >> 8) & 0xff;
    sg_BHeader[0x18] = (UCHAR)(m_Height >> 16) & 0xff;
    sg_BHeader[0x19] = (UCHAR)(m_Height >> 24) & 0xff;
    write(bmp， sg_BHeader， sizeof(sg_BHeader));
    for(i = 0; i < m_Height; i++)
    {
        unsigned short *c = p + (m_Height - 1 - i) * m_Width;
        unsigned short cc;
        for(j = 0; j < m_Width; j++)
        {
            cc = RGB565TO1555(*(c + j));
            write(bmp， &cc， 2);
        }
    }
close(bmp);
}

int main(int argc， char *argv[])
{
    unsigned char buf[SCREEN_WIDTH*SCREEN_HIGH*2];
    char *filename = "screenshot.bmp";
    int fb;

    if(argc != 5)
    {
        printf("usage: ./raw_bmp /dev/fb0 width high screen.bmp");
        return 0;
    }
    fb = open(argv[1]， O_RDONLY);
    if(fb < 0)
        exit(1);
    printf("reading screen...\n");
    read(fb， buf， SCREEN_WIDTH*SCREEN_HIGH*2);
    close(fb);
    printf("saving screen...\n");
```

```
    SaveBMPFile(buf， filename， atoi(argv[2])， atoi(argv[3]));
    printf("file %s created successfully\n"， filename);
    exit(0);
}
```

将上述源码保存为 raw_bmp.c，用如下方法交叉编译即可得到在目标板上运行的程序：

```
$ arm-linux-gcc -o raw_bmp raw_bmp.c
```

生成：raw_bmp，拷贝到目标板上运行：

```
$./raw_bmp /dev/fb0 480 272 screen.bmp
```

即生成截屏图片 screen.bmp。本程序源码在软件包目录 Helper2416/source/tests/raw_bmp.c

8.7 开发一个 QTOPIA 应用程序--目标板 LED 控制

QT / QTOPIA 都有自带的界面设计器 designer，使用该工具可以所见即所得的设计图形应用程序界面。在已经编译好的 qtopia-2.2.0/qt2/bin/designer 目录下（一定要编译过的，否则会出现找不到库等错误，编译 qtopia2.2 请参见 4.6 节）。

开发板上有 1 个可控制 LED 灯，下面一步一步介绍开发 LED 控制软件的开发过程。

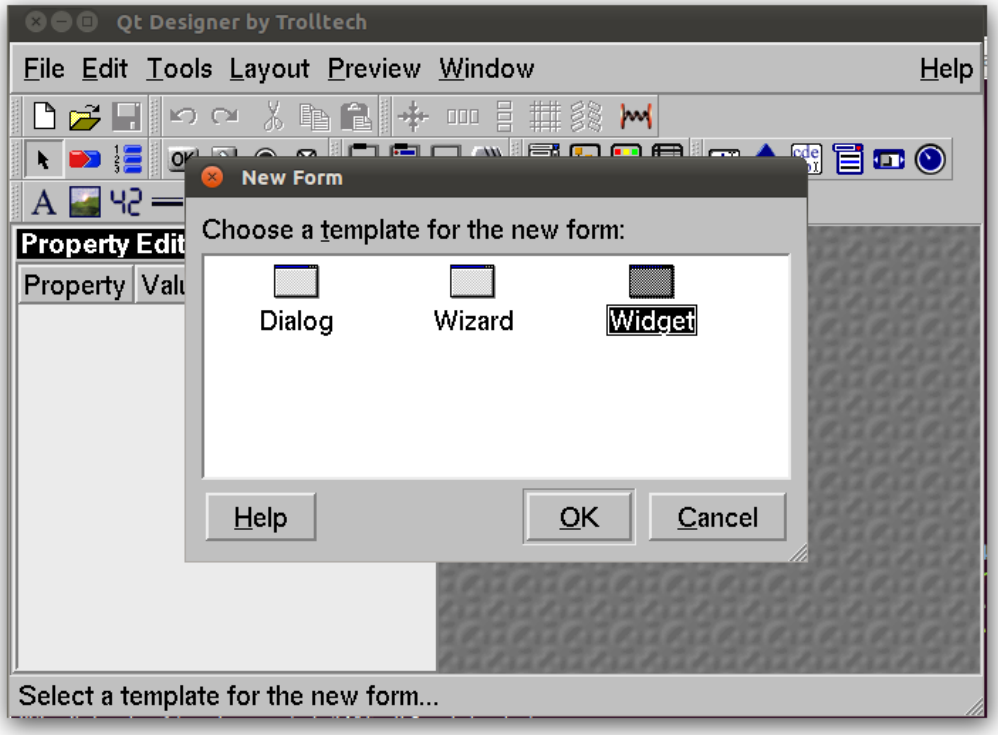
先创建一个工程目录，led_controller

```
$ mkdir led_controller
$ cd led_controller
```

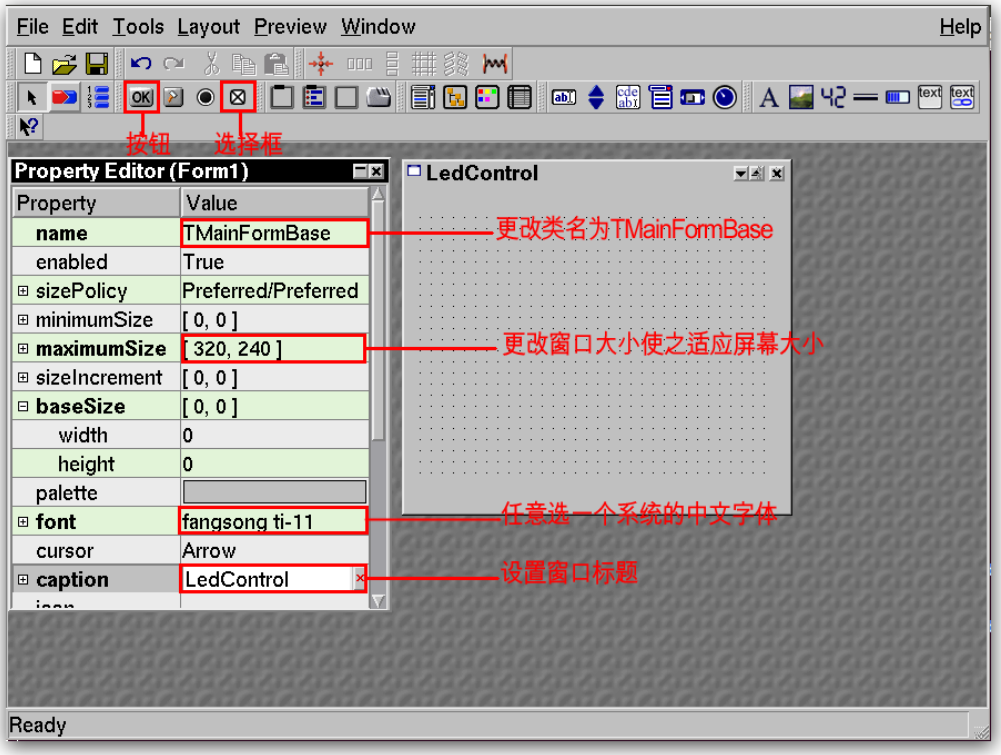
然后设置环境变量，启动 designer，设计 led_controller 界面，取名 led_controller.ui

```
$ source ~/workspace/Helper2416/arm-qtopia/qtopia-2.2.0/setQt2Env
$ designer led_controller.ui
```

选择 widget，如下图：

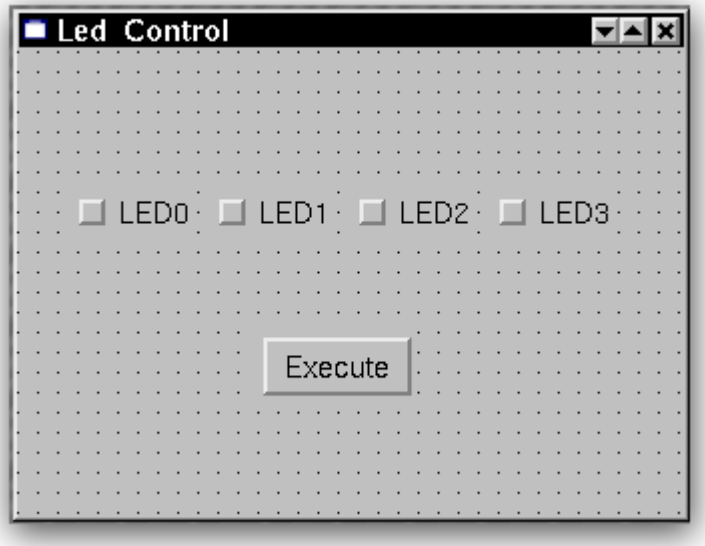


按 OK 后出现如下窗口：

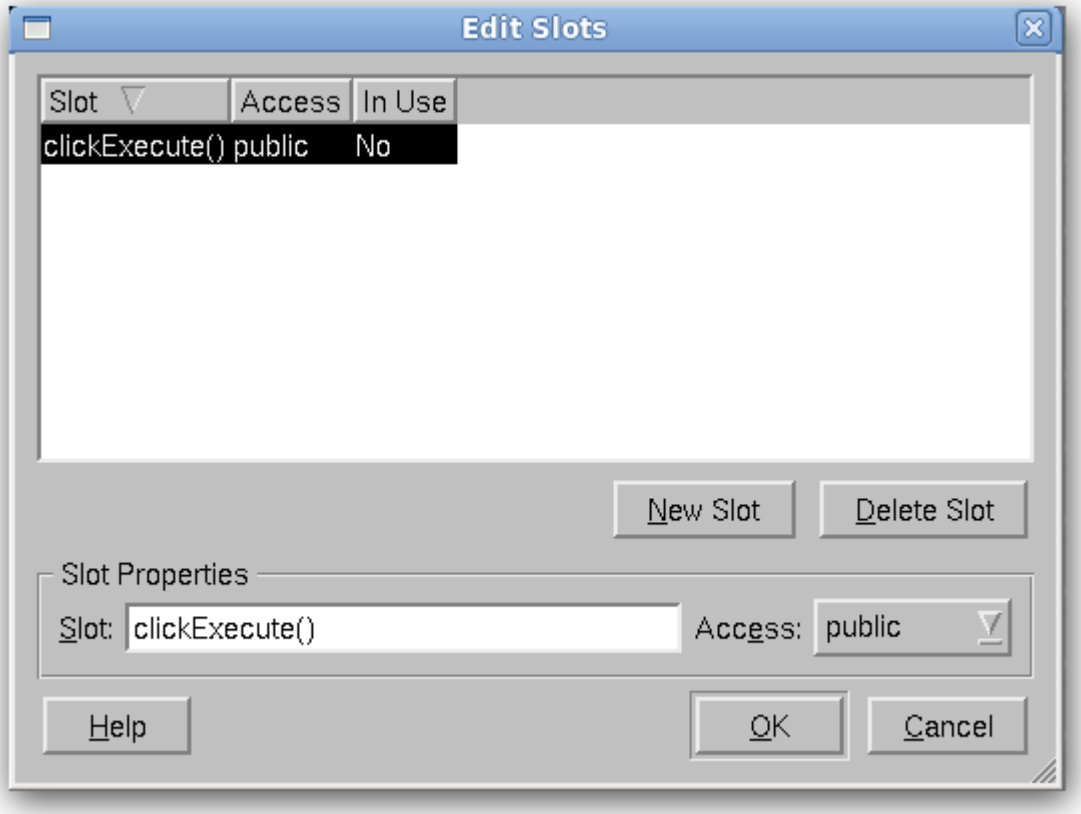


将原来的 Form1 类名更改为 TMainFormBase ，maximumSize 为 320，240，font 改为可支持中文的 fongsong ti-11，窗口标题更改为 LedControl，即是上图效果了。

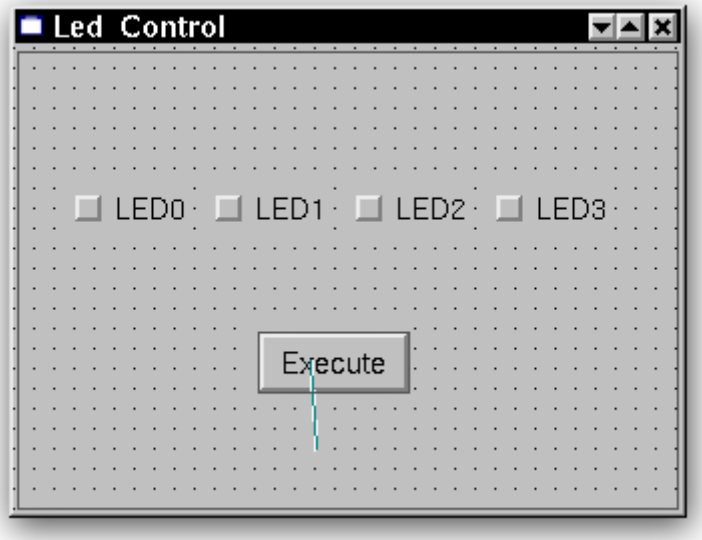
然后增加 4 个选择框和一个按钮，如下图：



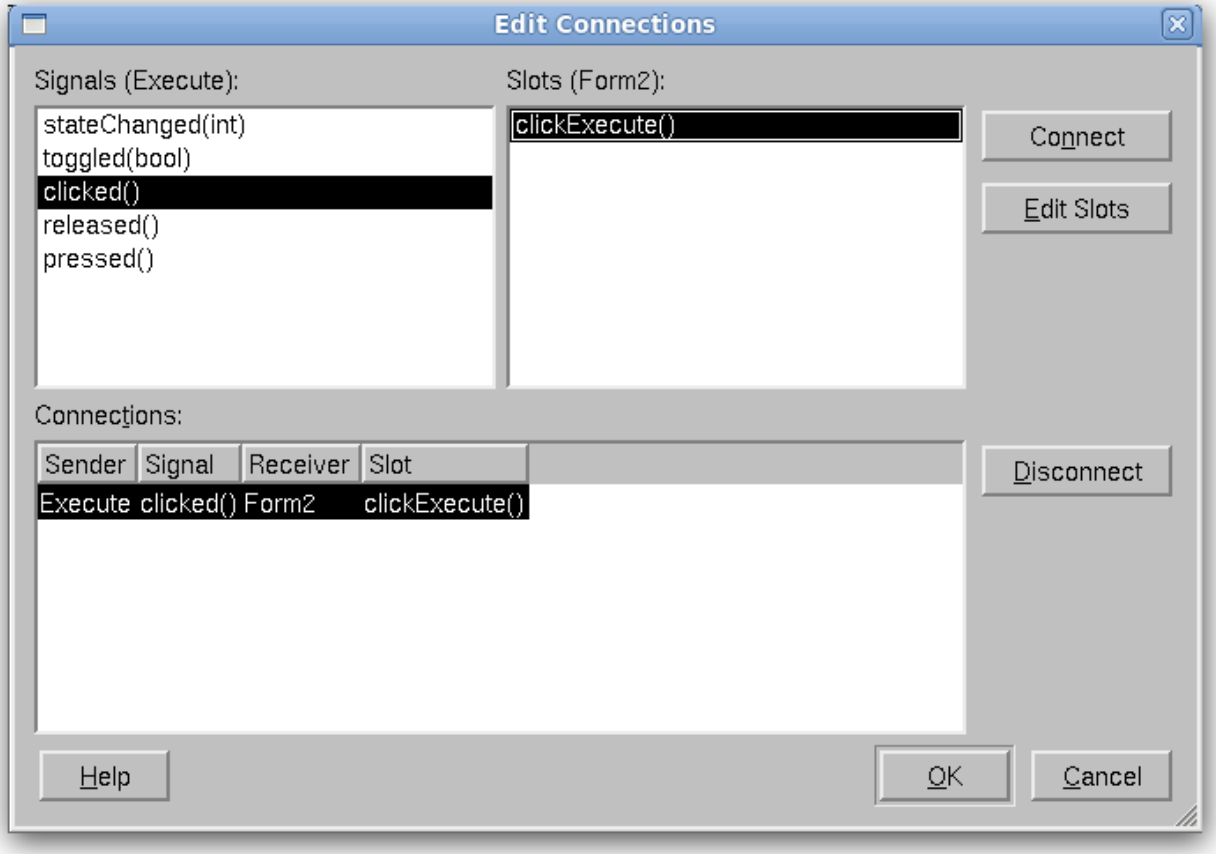
点击 Edit 菜单，选择 Slots，点击 New Slot，新建一个 Slot，输入一个函数名，如下图：



确定后，再点击 Tools 菜单，选择 Connect Signal/Slots，再点 Execute 按钮不放，拖到空白处，如图下：



弹出窗口：



左边选择 clicked()，表示点击事件，右边选择刚才创建的 slot：clockExecute()，即在下边生成一个连接，表示点击按钮会执行 clickExecute()函数。

按 OK 并保存退出，这样界面文件就创建完成了，下边编写代码。

在工程目录下创建 main.cpp，为 QT 运行主程序

```
#include "main_form.h"
#include <qtopia/qpeapplication.h>

QTOPIA_ADD_APPLICATION("led_control",TMainForm)
QTOPIA_MAIN
```

这个程序主要是两个宏定义，展开后是一个 main 函数，前边的 main_form.h 包含文件是与前面创建的界面文件关联的，创建之，内容如下：

```
#if !defined (__MAIN_FORM_H__)
# define __MAIN_FORM_H__

#include "led_form.h"
#include <qsocketnotifier.h>

class TMainForm: public TMainFormBase
{
    Q_OBJECT
public:
    TMainForm(QWidget * parent = 0, const char * name = 0, WFlags f =
WType_TopLevel) :
    TMainFormBase(parent,name,f) {}
    virtual ~TMainForm() {}

public slots:
    void ExecClicked();
};
#endif
```

该文件包含了 led_form.h，这个是 QT 编译器自动生成的，无需编写。

再编写窗口程序响应代码取名 main_form.cpp，内容如下：

```
#include "stdlib.h"
#include "stdio.h"
#include "unistd.h"
#include "main_form.h"
#include <qlineedit.h>
#include <qcheckbox.h>
#include <qglobal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define         LED_COUNT  4

void TMainForm::ExecClicked()
{
        QCheckBox* ChkLed[LED_COUNT] = {ChkLed0, ChkLed1, ChkLed2,
ChkLed3};
        int         i;
```

```
        FILE*       fdset[LED_COUNT];
        const char *path[LED_COUNT] = {"/sys/class/leds/led4/brightness",
                                       "/sys/class/leds/led5/brightness",
                                       "/sys/class/leds/led6/brightness",
                                       "/sys/class/leds/led7/brightness"};
        for(i=0; i<LED_COUNT; i++)
        {
                fdset[i] = fopen(path[i], "w+");
                if(!fdset[i])
                {
                        printf("%s: error open file\n", __FUNCTION__);
                        exit(0);
                }

        }

        for(i=0; i<LED_COUNT; i++)
        {       if(ChkLed[i]->isChecked())
                {
                        fwrite("1", 1, 1, fdset[i]);
                        fflush(fdset[i]);
                } else
                {
                        char    cmd[256];
                        sprintf(cmd, "/bin/echo 0 > %s", path[i]);
                        system(cmd);    // same as file operation, just a demostration
                }
        }
}
```

代码很简单，就是通过之前创建的 slot 响应按钮操作，然后就是根据选项写设备文件，让驱动去工作，控制 LED 灯亮或者灭。

下面创建 QT 工程文件：

在程序目录下运行：

```
$ qmake -project
```

就会生成一下 led_control.pro 的 QT 工程文件，内容如下：

```
#############################################################################
# Automatically generated by qmake (2.01a) ?? 1? 13 13:45:30 2012
#############################################################################

TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += main_form.h
FORMS += led_form.ui
```

```
SOURCES += main.cpp main_form.cpp
```

编辑该文件：

中间加两行，并写上 TARGET 名称：

```
#############################################################################
# Automatically generated by qmake (2.01a) ?? 1? 13 13:45:30 2012
#############################################################################

CONFIG          += qtopiaapp
CONFIG          -= buildQuicklaunch

TEMPLATE = app
TARGET = led_control
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += main_form.h
FORMS += led_form.ui
SOURCES += main.cpp main_form.cpp
```

上面红色的是增加的项目，CONFIG+=qtopiaapp 表示要编译的是一个 Qtopia 程序，CONFIG-=buildQuicklaunch 的意思是需要编译成一个独立的可执行程序，而不是应用程序插件，而 TARGET 则指定最后生成的可执行文件名为 led_control。

代码都写完了，下面创建基本目标板平台的编译脚本 buildarm.sh，内容如下：

```
#!/bin/sh

source ~/workspace/Helper2416/arm-qtopia/qtopia-2.2.0/setQpeEnv
qmake -spec qws/linux-arm-g++ -o Makefile.target *.pro
make -f Makefile.target clean
make -f Makefile.target
```

给脚本增加执行属性，并执行：

```
$ chmod +x buildarm.sh
$ ./buildarm.sh
```

会生成一个 Makefile.target，并自动进行编译，最后生成 led_control，这个文件是可执行文件，可以放到目标板运行。

按照第 5 章介绍的方法拷贝 led_control 到目标板，按照如下方法运行，就可以运行看效果了。

将文件拷贝到目标板任意目录，假设是根目录后，先运行 qt 环境脚本，在我们的目标板文件系统里已经放置了这个脚本：

```
$ source qtenv
```

注意：source 命令需要 bash 才能解释执行，目标板上已经提供了 bash，需要登录进目标板才运

行，比如用 root 登录。

然后直接执行 led_control：

```
$ ./led_control
```

在屏幕上就会出现程序界面了。按照 2.2.5 节的方法测试就行了，够简单了吧。

在软件包里有完整源码：Helper2416\source\tests\led_control。

8.8 关于部署 LINUX 开发环境的建议

对于个人，建议采用虚拟机安装 LINUX 的方式，省钱也方便，如果觉得进出虚拟机很麻烦，则可以使用远程桌面的方式登录到虚拟机操作，所有的操作都在 Windows 的一个窗口里进行，可以使用文本复制、粘贴等操作(不能拷贝文件)。共享文件则可以用 Linux 的 samba 功能，类似 Windows 文件共享方式。具体的操作方法自行上网搜索。

对于企业，多人协作开发的团队，建议集中安装 LINUX 服务器，可以配一台性能好一点的机器，多人通过远程桌面方式登录服务器同时开发，方便编译器、根文件系统、共享库文件等集中管理，可大大提高开发效率，同时也结省了硬件成本。文件共享采用 Linux 的 samba、ssh，nfs 等方式。

以上远程桌面登录软件很多，有 x-win32，exceed，Xmanager 等，根据多年的工作经验，建议采用 x-win32(8.1.1122 版本比较稳定)作为远程登录工具，效率最高，远程操作的速度接近本地操作的速度。

我们可以为企业客户提供 LINUX 服务器开发环境搭建等服务。

8.9 推荐几个好网站：

http://mirrors.163.com

http://mirrors.sohu.com

163 和 sohu 提供的国内的 LINUX 镜像服务器，下载速度飞快，里边有多种 LINUX 发行版。

http://www.rpmfind.net

当你不知道哪个软件包里有你需要的软件时，通过她查就对了，当然，这个服务器经常会挡机，不行就多试几次。

http://git.oschina.net/progit/

一个介绍 GIT 使用的网站，各种语言都有，介绍完整，非常好用。

注：GIT 是 LINUX 之父编写版本管理工具，开源、免费、而且功能强大。

8.10 关于技术支持

对于所有购买本公司开发板的用户，您可以通过以下方式获得永久免费技术支持（限开发板上已经提供的功能）：

邮件：support@jyxtec.com

论坛：http://bbs.eeworld.com.cn/forum-220-1.html

电话：0755-86017779

手机：13510178868

QQ 群：74306575

需要技术支持请在工作时间联系。

8.11 关于定制服务

本公司提供基于 S3C2416 平台项目的定制服务，如有需要请联系：13510178868　陈先生。