



OK6410-A

Linux 使用手册

Devoted to create the best embedded products

www.witech.com.cn

www.forlinx.com

www.helloarm.com

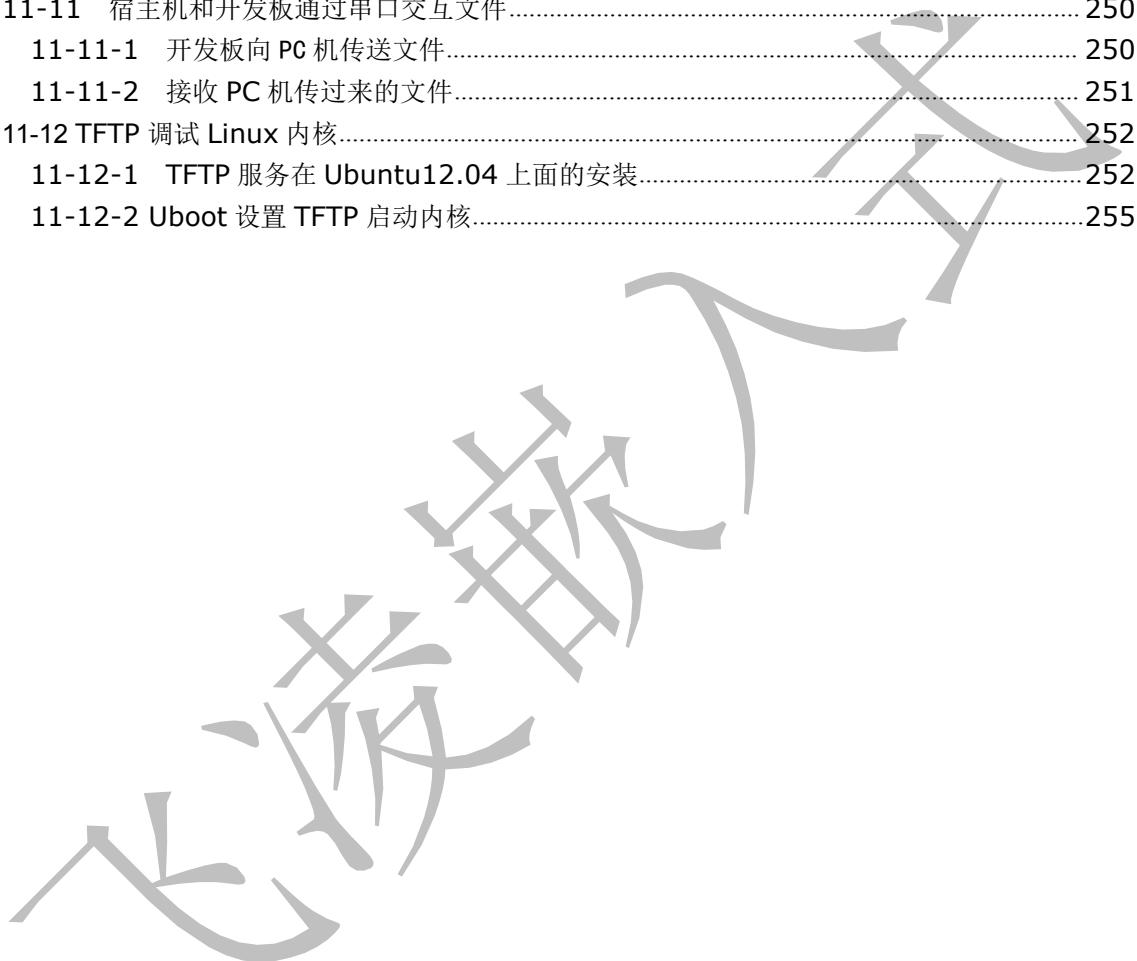
目 录

第一章 前言	8
第二章 一键烧写 LINUX.....	9
2-1 对于 WinCE 系统换 Linux 系统的特殊说明.....	11
2-1-1 Eboot 擦除 NandFlash 方法.....	11
2-1-2 Linux 擦除 NandFlash 的方法.....	12
2-2 制作用于一键烧写 Linux 的 SD 卡.....	14
2-3 烧写 Linux 到开发板的 NandFlash 中.....	20
2-4 出现坏块怎么办?	25
第三章 USB 烧写 LINUX.....	28
3-1 制作用于 USB 烧写 Linux 的 SD 卡.....	29
3-2 USB 烧写 Linux 到开发板的 NandFlash.....	34
3-2-1 设置开发板从 SD 卡启动.....	34
3-2-2 下载并烧写 u-boot 到 Nandflash.....	36
3-2-3 烧写 Kernel.....	39
3-2-4 烧写根文件系统.....	41
3-3 安装 USB 下载专用驱动.....	41
3-4 设置开发板从 Nandflash 启动.....	45
第四章 测试 LINUX-3.0.1.....	47
4-1 Qttopia2.2.0、QT-Extended-4.4.3、QT/E-4.7.1 切换.....	47
4-2 触摸屏的校准与重新校准.....	48
4-3 修改 LCD 分辨率.....	49
4-4 SD/MMC 卡驱动测试.....	50
4-5 USB 鼠标输入和触摸输入的切换.....	52
4-6 USB HOST 接口测试.....	53
4-6-1 USB 鼠标.....	53
4-6-2 USB 键盘.....	54
4-6-3 挂载 U 盘.....	55
4-7 以太网驱动测试及相关服务.....	57
4-7-1 网络相关配置.....	57
4-7-2 使用 ping 命令测试网络.....	60
4-7-3 浏览网页.....	60
4-7-4 Telnet 服务.....	61
4-7-5 FTP 服务.....	62
4-7-6 Web 服务.....	62
4-7-7 NFS 挂载网络文件系统.....	63

4-8 音频播放测试.....	67
4-9 视频播放测试.....	69
4-10 录音测试.....	71
4-10-1 语音记事本.....	71
4-10-2 ben 测试程序.....	72
4-11 采样测试.....	72
4-12 LED 测试.....	74
4-13 蜂鸣器测试.....	75
4-14 TV 输出.....	76
4-15 红外测试.....	76
4-16 温度传感器测试.....	76
4-17 串口测试.....	77
第五章 测试外围模块.....	79
5-1 USB 3G 上网卡测试.....	79
5-2 CMOS 摄像头 OV9650 测试.....	82
5-3 SDIO WIFI.....	83
5-4 串口扩展板.....	86
5-5 4X4 和 8X8 矩阵键盘.....	87
5-6 USB 摄像头.....	88
5-7 485 接口.....	88
5-8 CAN 转接板.....	88
5-9 GPRS 模块测试.....	89
第六章 在主机上搭建 LINUX 开发环境.....	96
6-1 安装 Ubuntu12.04.....	96
6-2 将 Ubuntu 设置为 root 用户自动登录.....	103
6-3 设置 Ubuntu 网络参数.....	104
6-4 Ubuntu 常用应用程序.....	109
6-4-1 Linux 终端.....	109
6-5 安装交叉编译器.....	111
第七章 编译 UBOOT 和 LINUX 内核.....	113
7-1 编译 u-boot-1.1.6.....	114
7-1-1 编译 128M 内存开发板 Uboot 方法:	114
7-1-2 编译 256M 内存开发板 Uboot 方法:	116
7-2 编译 Linux-3.0.1.....	117
7-2-1 配置内核.....	117
7-2-2 编译内核.....	117
7-2-3 开发板驱动源码路径.....	118
第八章 制作 YAFFS2 文件系统映像.....	120
8-1 准备好文件系统.....	120

8-2 制作映像.....	120
第九章 多媒体硬件编解码测试.....	121
9-1 多媒体功能简介.....	121
9-2 多媒体综合测试一.....	121
9-2-1 编译 MultimediaTest 程序.....	121
9-2-2 H.264 解码.....	124
9-2-3 MPEG4 解码.....	126
9-2-4 H.263 解码.....	127
9-2-5 VC-1 解码.....	128
9-2-6 多种视频同时解码.....	129
9-2-7 H.264 解码&LCD 双缓冲.....	130
9-2-8 摄像头预览&MFC 编码.....	131
9-2-9 H.264 解码&摄像头预览.....	133
9-2-10 摄像头预览&MFC 编码解码.....	135
9-2-11 摄像头预览&JPEG 编码.....	137
9-2-12 JPEG 解码.....	138
9-3 多媒体综合测试二.....	140
9-3-1 命令行的开源播放软件 player.....	140
9-3-2 Qt 图形界面程序 player-qt4.....	144
9-3-3 如何转换普通的多媒体文件到 6410 支持播放格式.....	146
第十章 LINUX 基础实验教程.....	147
10-1 实验一 shell 编程.....	147
10-2 实验二 Hello world.....	151
10-3 实验三 多线程实验.....	153
10-4 实验四 多进程实验.....	158
10-5 实验五 网络编程实验—服务器/客户机.....	165
10-6 实验六 Makefile 实验.....	182
10-7 实验七 进程间通讯.....	188
第十一章 附录.....	206
11-1 Windows XP 系统中使用虚拟机搭建开发环境.....	206
11-1-1 在 Windows XP 中安装 VMware Workstation.....	206
11-1-2 VMware 新建并设置 Ubuntu 安装环境.....	206
11-1-3 VMWARE-TOOLS 的安装.....	212
11-2 使用 FTP 在 XP 和 Ubuntu 间传输文件.....	213
11-2-1 设置 Ubuntu 网络参数.....	213
11-2-2 安装 Ubuntu 的 vsftpd 服务.....	214
11-2-3 安装 Windows XP 的 FTP 客户端工具.....	216
11-3 Ubuntu 中使用 dnw 下载.....	221
11-4 Windows 超级终端使用说明.....	221

11-5 字符设备驱动架构分析.....	223
11-6 Minicom 使用指南.....	228
11-6-1 minicom 介绍和设置.....	228
11-6-2 Ubuntu12.04 的 minicom.....	232
11-7 Linux 常用命令详解.....	241
11-8 烧写用 SD 卡和 SD 读卡器的问答.....	244
11-9 Dnw 软件的使用简便教程.....	245
11-10 Qt4.7 支持中文显示.....	248
11-11 宿主机和开发板通过串口交互文件.....	250
11-11-1 开发板向 PC 机传送文件.....	250
11-11-2 接收 PC 机传过来的文件.....	251
11-12 TFTP 调试 Linux 内核.....	252
11-12-1 TFTP 服务在 Ubuntu12.04 上面的安装.....	252
11-12-2 Uboot 设置 TFTP 启动内核.....	255



版本号	版本说明
Linux3.0.1-2013-01 版本说明	<p>1、增加对 MT29F8G08ABABA (SLC) 1G NandFlash 的支持。</p> <p>2、发现 sdiowifi 在某些特定情况下会造成连接出错，或连接断开的问题。</p>
Linux3.0.1-2012-09 版本说明	<p>1 增加对 K9LBG08U0D (MLC) 4G NandFlash 的支持。</p> <p>2 Uboot 增加启动菜单功能，方便用户设置相关参数。</p> <p>3 Uboot 增加网络支持，这样就可以使用 tftp 功能加载内核，方便用户调试 Linux 驱动。</p> <p>4 增加 USB 摄像头测试程序。</p>
Linux3.0.1-2012-05 版本说明	<p>1.增加 Qt 图形界面测试程序，新增 串口测试,GPRS 测试(可以拨打,接听电话,收发短信,拨号上网)</p> <p>2.增加支持 3G 上网卡类型，目前支持： WCDMA: AD3812 中兴模块，飞凌自产 CDMA2000: AC581,AC582 中兴模块 TDSCDMA: ET127 (华为),A356 (中兴)</p> <p>3.增加温度传感器 18b20 驱动及测试程序,红外驱动及测试程序。</p> <p>4.解决 AD 转换与触摸屏资源冲突问题，触摸功能和 AD 可调节电阻测试功能同时使用。</p> <p>5.解决矩阵键盘驱动，PWM 驱动存在的 Bug。</p> <p>6.解决音频录音功能，可使用 Qtopia 语音记事本实现录音，播放录音。</p>
Linux3.0.1-2012-02 版本说明	<p>1.支持一键烧写 yaffs2 文件系统，烧写进度可在 LCD 上显示(目前支持 4.3 寸屏)，不再需要 PC 显示。</p> <p>2.添加制作 yaffs2 文件系统映像的方法</p> <p>3.Qtopia 新增网络 IP 地址设置应用程序。</p> <p>4.修正了多媒体测试程序中关于摄像头 H.264 编解码相关的 Bug 和 Linux 内核中 V4L2 驱动相关 Bug。</p> <p>5.多媒体中修正 JPEG 编解码的 Bug。</p> <p>6.手册中增加 NFS 挂载网络文件系统的方法。</p>

Linux3.0.1- 2011-11 版本说明

1. 移植 Linux3.0.1 到 OK/TE6410 开发板
2. 添加 CAN、485、USB 摄像头、TV 输出、矩阵键盘等基本的驱动到 Linux 内核代码中。
3. 完善串口扩展板、LCD 分辨率设置的资料。
4. 手册增加 NFS 挂载宿主机目录的使用方法。

SLC 和 MLC 的区别:

存储单元分为两类: SLC (Single Level Cell 单层单元) 和 MLC (Multi-Level Cell 多层单元)。此外, SLC 闪存的优点是复写次数高达 100000 次, 比 MLC 闪存高 10 倍。此外, 为了保证 MLC 的寿命, 控制芯片都校验和智能磨损平衡技术算法, 使得每个 存储单元 的写入次数可以平均分摊, 达到 100 万小时故障间隔时间(MTBF)。

1、资料唯一更新方式:

<http://bbs.witech.com.cn/thread-3809-1-1.html>

2、在您的使用过程中如果遇到相关技术问题, 欢迎访问飞凌官方论坛寻求答案, 或者发帖咨询。论坛地址:

<http://bbs.witech.com.cn/>

3、飞凌技术服务热线: 0312-3119192

4、本手册版权归属飞凌嵌入式有限公司所有, 并保留一切权利。
任何单位及个人不得擅自摘录本手册部分或全部内容。

5、本手册重点讲述嵌入式 Linux 的开发与教程, 如果您对开发板的硬件不熟悉, 或者对开发板接口不熟悉, 强烈建议把飞凌 6410 硬件手册通读一遍。手册中如果遇到不明白的硬件术语, 请查阅飞凌 6410 硬件手册。

本文会详细的叙述操作过程。不再针对某些 Linux 知识作详细叙述。如果您对 Linux 系统的使用和命令不熟悉, 可以在

<http://bbs.chinaunix.net/> 等国内有规模的 Linux 论坛中查询。



第一章 前言

感谢您使用‘飞凌嵌入式’的产品！

飞凌公司从 06 年成立至今已为客户提供数万套的开发平台，致力于帮助初学者顺利入门，并为工程师的产品设计提供参考，我们专注于嵌入式开发平台的构造、搭建、移植和升级服务，努力使嵌入式系统的学习和产品开发更容易更简单。

Linux 是嵌入式系统学习的首选，它本身是个开源的项目，可以使学习者通过研究和实践逐渐领会嵌入式操作系统的实质，同时，它又是一个成功的软件平台，在各行各业已经有大量应用，可以这样讲，不懂 **Linux** 的话将不是一个合格的嵌入式软件工程师。

嵌入式系统的发展也表现在硬件技术的不断更新，两年前，**ARM9** 还是工程师们产品设计的首选平台，而现在，**ARM11** 却开始崭露头角，相信在以后几年内，它的应用也将越来越多，成为嵌入式技术学习和开发的主角。

S3C6410 是基于 **ARM11** 内核来设计的，它相对于 **ARM9** 不仅仅是速度性能的提升那么简单，而在其他先进功能上更具学习开发的价值，例如，**S3C6410** 内部集成了视频流编解码的功能，工程师可以对照研究其工作机理；2D/3D 加速的应用也可以使学习者尝试这方面的探索；另外，只有在 **S3C6410** 上才可以完美运行 **Android** 等充满潜力的操作系统，当然，**S3C6410** 还具有先进的 OTG 接口，能支持 **SLC/MLC** 等主流的 **NAND FLASH**。

OK6410，是一款性价比和配置很高的学习开发套件，我们为推出该套件做了精心的准备，外围接口十分丰富，另外还专门配备了多个扩展模块，**CMOS** 摄像头模块、**WIFI** 无线模块等供大家选择。

该手册主要讲述在 **OK6410** 开发板上 **Linux** 系统的构建和开发，由于篇幅有限，关于 **Linux** 本身基本操作类的知识没有过多介绍，这方面资料较多，大家可以结合更专业的 **Linux** 书籍或其他光盘资料配合学习。

到目前为止，手册虽然经过多次修改，但仍有很多不足，请大家多提宝贵意见！同时，手册也在不断更新和充实当中，我们会把更新的内容放到网站客户服务专区，供大家下载。

最后，预祝大家快乐的学习和工作！

飞凌嵌入式 •2012

第二章 一键烧写 Linux

● 什么是一键烧写 Linux？有什么用？

简单的说，就是借助 SD 卡、飞凌提供的程序和系统映像，通过一系列操作，非常迅速的烧写 Linux 到开发板的 NandFlash 中。

您可以用一键烧写解决什么样的问题呢？比如说：

1. 开发板在出厂时默认烧写 WinCE6.0 系统。如果您希望 OK6410 畅快淋漓的运行 Linux 系统，就需要把 Linux 重新烧写开发板的 NandFlash 中。
2. 更新 uboot、内核 (zImage)、文件系统 (Yaffs2 文件系统) 中的一个或者多个。
3. 由于各种原因造成的开发板无法启动。

注意： 一键烧写系统时请使用 SD 卡 + USB 读卡器方式，大家尽量不要使用 TF 卡 + MicroSD + 读卡器。另外，我们推荐使用 4G ScanDisk 或者 4G Kingston (金士顿) 的 SD 卡，当然您手头有其他类型的 SD 卡且能成功烧写系统也可以，如不能成功烧写系统，尽量使用我们推荐的 SD 卡。

● 一键烧写 Linux 与 USB 烧写 Linux 有什么异同？

相同点：1. 目的都是将 Linux 烧写在 NandFlash 中。所以，烧写方法任选其一，都可以完成 Linux 的烧写。

2. 基本方法都是借助 SD 卡启动。
3. 需要通过串口查看烧写的状态。

不同点：1. 一键烧写速度快，一次烧写必须烧写所有文件

USB 烧写速度慢，可以单个更新文件。

2. 一键烧写只需要借助 SD 卡
- USB 烧写除了 SD 卡，还需要 USB 线。

● 介绍完一键烧写 Linux 的用途后，开始介绍一键烧写 Linux 到开发板的详细过程

一键烧写 Linux 支持烧写 yaffs2 格式的根文件系统。

烧写之前，请先准备好一个 SD 卡和一个 SD 读卡器

(附录：SD卡和SD读卡器常见问题解答)

本章所需文件路径:

文件名 (文件用途)	文件在基础光盘中的路径
SD_Writer.exe(PC 烧写工具)	Linux-3.0.1\Linux 烧写工具\
mmc_ram128.bin (128M 内存开发板适用的 sdboot, 用于 sd 卡启动)	Linux-3.0.1\Linux 烧写工具\
mmc_ram256.bin (256M 内存开发板适用的 sdboot, 用于 sd 卡启动)	Linux-3.0.1\Linux 烧写工具\
u-boot_ram128.bin (uboot 映像, 适用于 128M 内存开发板)	Linux-3.0.1\demo\
u-boot_ram256.bin (uboot 映像, 适用于 256M 内存开发板)	Linux-3.0.1\demo\
zImage(内核映像)	Linux-3.0.1\demo\
rootfs.yaffs2-nand256m (用于触摸屏输入的 yaffs2 文件系统, 适用于 256M nandflash 的开发板)	Linux-3.0.1\filesystem\
rootfs.yaffs2-nand2g (用于触摸屏输入的 yaffs2 文件系统, 适用于 1G 字节或 2G 字节或者 4G 字节的 nandflash 的开发板)	Linux-3.0.1\filesystem\
DNW(串口、USB 口工具)	实用工具\

注意: 请您在制作 SD 卡一键烧写前一定要注意以下内容。

本手册中提供的 linux3.0.1 文件系统区分 nand flash 大小, UBOOT 和 mmc.bin 区分内存大小。具体配置和镜像的对应关系请参考上表。

例如:

u-boot_ram128.bin 是适用于 128M 内存开发板的。

2-1 对于WinCE系统换Linux系统的特殊说明

飞凌 6410 开发板预装的系统是 WinCE6.0, WinCE 系统在微软设计的时候,有一个约定,就是把前四个块都标记成了坏块!也就是说把 bootloader 分区都标记成坏块,以防止 bootloader 被 WinCE 应用程序擦掉。这样就带来一个问题,就是在开发板换 Linux 系统或者 Android 系统的时候,需要把这几个‘假坏块’恢复过来,可以使用 WinCE 系统的 Eboot 或者 Linux 的 mmc.bin 擦除‘假坏块’,擦除以后才可以烧写 Linux 或者 Android。

2-1-1 Eboot 擦除 NandFlash 方法

注意: 如果 Eboot 已经无法启动, 需要先烧写 Eboot 到开发板中。烧写 Eboot, 请参看“飞凌 6410 开发板 WinCE6.0 用户手册 (图文版).pdf”中的 3-1 章节到 3-3 章节。

步骤 1. 先用串口线连接好开发板 COM0 与 PC 机的串口, 打开并设置 DNW 软件 (附录:dnw软件的使用简便教程)。

步骤 2. 然后给开发板上电, 等到出现延时 5 秒启动系统时, 在 DNW 软件中按 PC 键盘的空格键使开发板停留在 Eboot 状态。

```
WinCE 6.0 Stepper for SMDK6410
Launch Eboot...

Microsoft Windows CE Bootloader Common Library Version 1.4 Built Feb 19 2011 10:26:13
Microsoft Windows CE Bootloader for the Samsung SMDK6410 Version 2.4 Built Mar 8 2011

+DALArgsInit()
SocID:0x36418101
Arguments area is initialized
-DALArgsInit()
INFO: (unsigned)C_IsrHandler : 0x800402D4
INFO: (unsigned)ASM_IsrHandler : 0x800428E4
INFO: (unsigned)pISR : 0xE010A31
BP_Init
[FMD] ++FMD_Init() ****
[FMD:INF] FMD_Init() : Read ID = 0x0000ecd3
[FMD] FMD_Init() : NUM_OF_BLOCKS = 4096
[FMD] FMD_Init() : PAGES_PER_BLOCK = 128
[FMD] FMD_Init() : SECTORS_PER_PAGE = 4
[FMD] --FMD_Init()
[FMD] FMD_GetInfo() : NUMBLOCKS = 4096(0x1000), SECTORSPERBLOCK = 128(0x80), BYTESPERSECTOR = 2048(0x800)
[FMD] FMD_GetInfo() : NUMBLOCKS = 4096(0x1000), SECTORSPERBLOCK = 128(0x80), BYTESPERSECTOR = 2048(0x800)
wNUM_BLOCKS : 4096(0x1000)
[Eboot] ++InitializeDisplay()
[Eboot] --InitializeDisplay()
Press [ENTER] to launch image stored on boot media, or [SPACE] to enter boot monitor.

Initiating image launch in 5 seconds. 这里是 5 秒延时
```

步骤 3. 按下‘A’键擦除 NandFlash。如图：

```
Initiating image launch in 5 seconds.

Ethernet Boot Loader Configuration:

0) IP address: 0.0.0.0
1) Subnet mask: 255.255.255.0
2) DHCP: Disabled
3) Boot delay: 5 seconds
4) Reset to factory default configuration
5) Startup image: LAUNCH EXISTING
6) Program disk image into SmartMedia card: Enabled
7) Program DM9000A MAC address (00:00:00:00:00:00)
8) KITL Configuration: DISABLED
9) Format Boot Media for BinFS

A) Erase All Blocks
B) Mark Bad Block at Reserved Block
C) Clean Boot Option: FALSE
D) Download image now
E) Erase Reserved Block
F) Low-level format the Smart Media card
L) LAUNCH existing Boot Media image
R) Read Configuration
S) Lcd Resolution select(800x480)
U) DOWNLOAD image now(USB)
W) Write Configuration Right Now

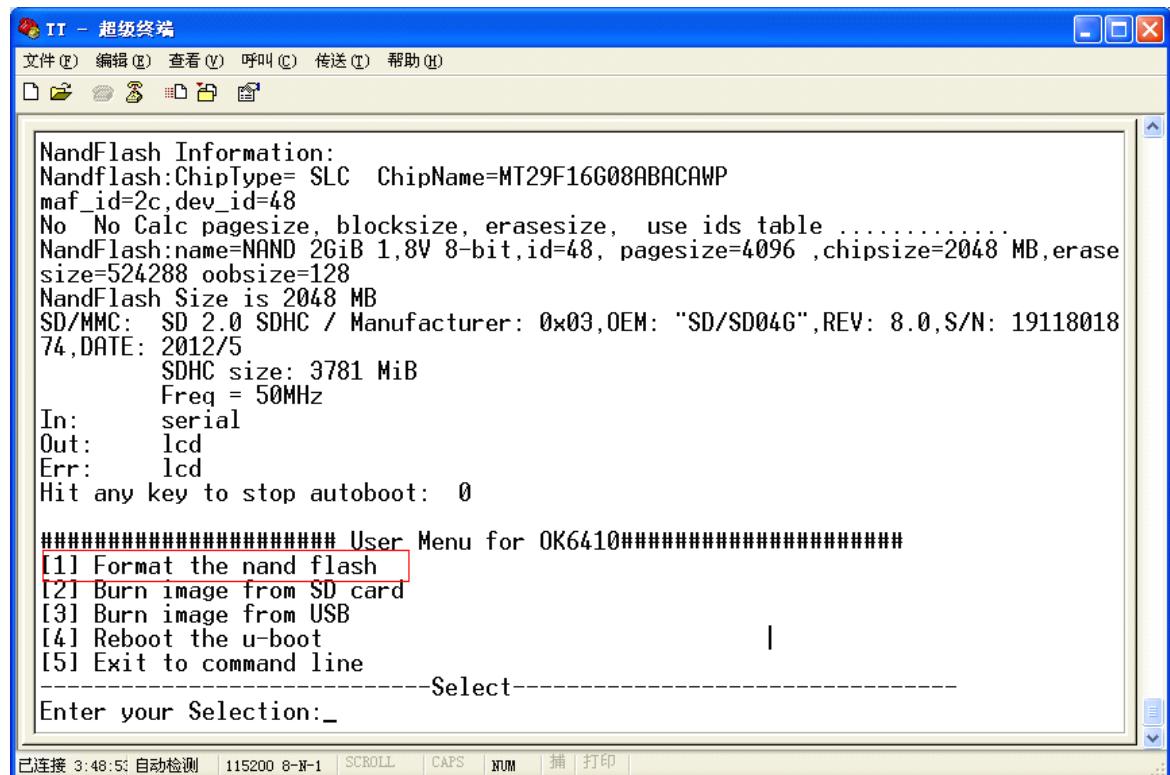
Enter your selection:
```

经过这一小节的操作后，就可以开始烧写 Linux 系统了。

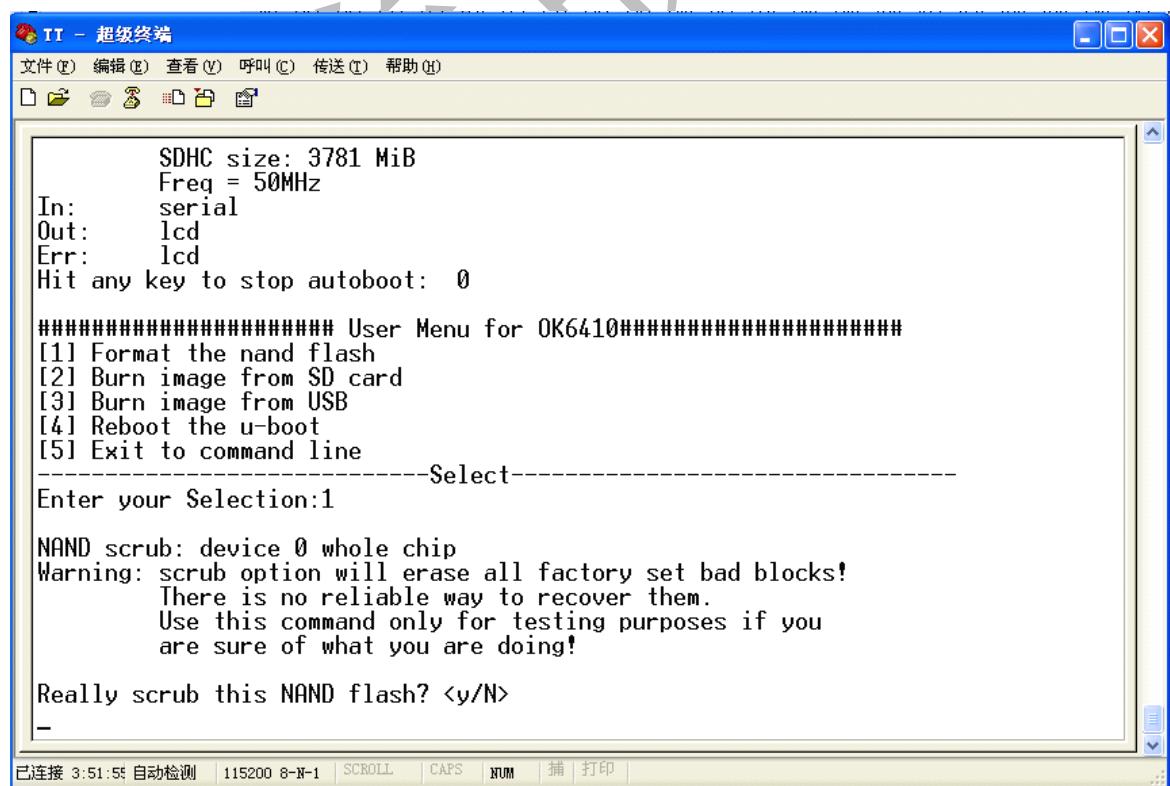
2-1-2 Linux 擦除 NandFlash 的方法

如果您不小心忘记了使用 WinCE 的 Eboot 擦写 NandFlash，就可以使用如下的方法完成 WinCE 到 Linux 的切换了。

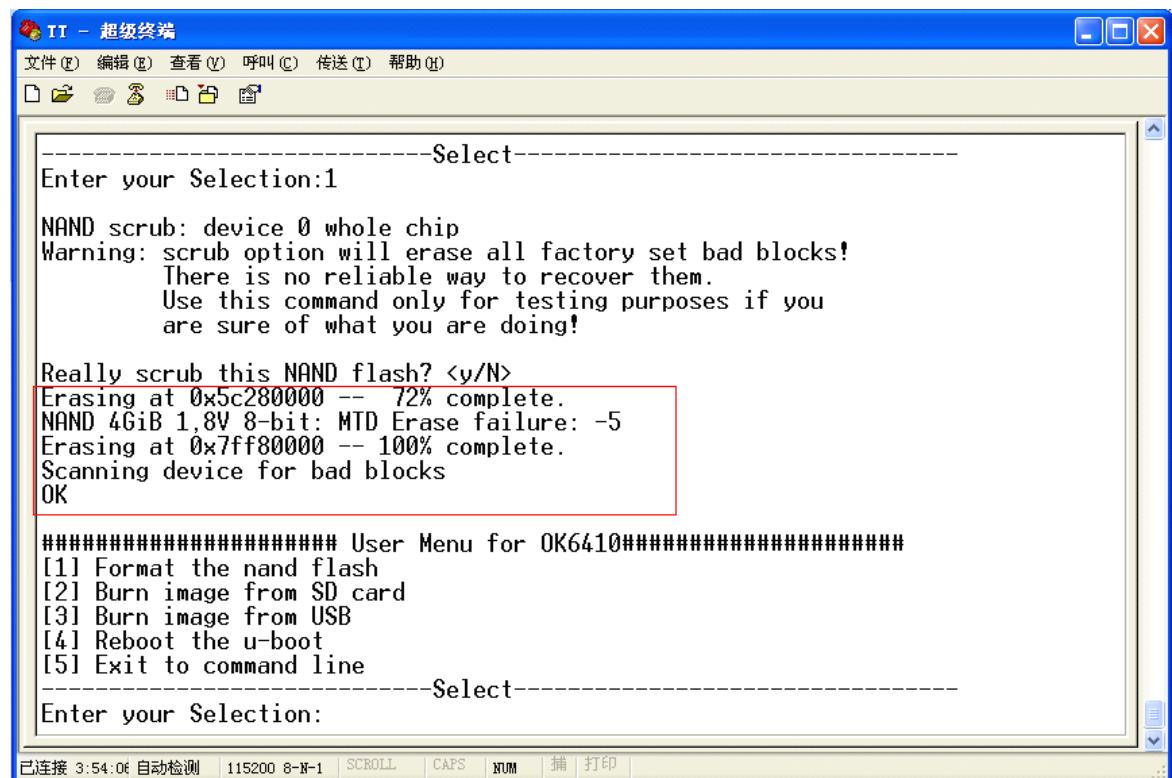
首先按照 [2-3 烧写 Linux 到开发板的 NandFlash 中](#)一节的方法制作一键烧写的 SD 卡。把 SD 卡插入到开发板中，拨码开关设置为 SD 卡启动方式，调试串口跟 PC 串口连接，开机按 PC 空格键会进入 mmc.bin 的菜单选择状态，如下图所示：按照下面的步骤即可完成擦除任务。



输入 1，代表执行菜单 1 命令：



输入 y, 回车, 即可擦除 NandFlash。



可以看到, 已经擦除了整个 Nand, 这里有一个坏块, 不过不影响使用的, 无论 MLC 的 NandFlash 还是 SLC 的 NandFlash, 允许有一定的坏块存在, 接下来重新启动开发板, 就开始 Linux 的安装之旅了。

2-2 制作用于一键烧写Linux的SD卡

步骤 1: 将 SD 卡格式化为 FAT32 格式

将 SD 卡接入 SD 读卡器中, 把 SD 读卡器插在 PC 机的 USB 口中。



等到 PC 机能够正常识别出 SD 卡，把 SD 卡格式化为 FAT32 格式。

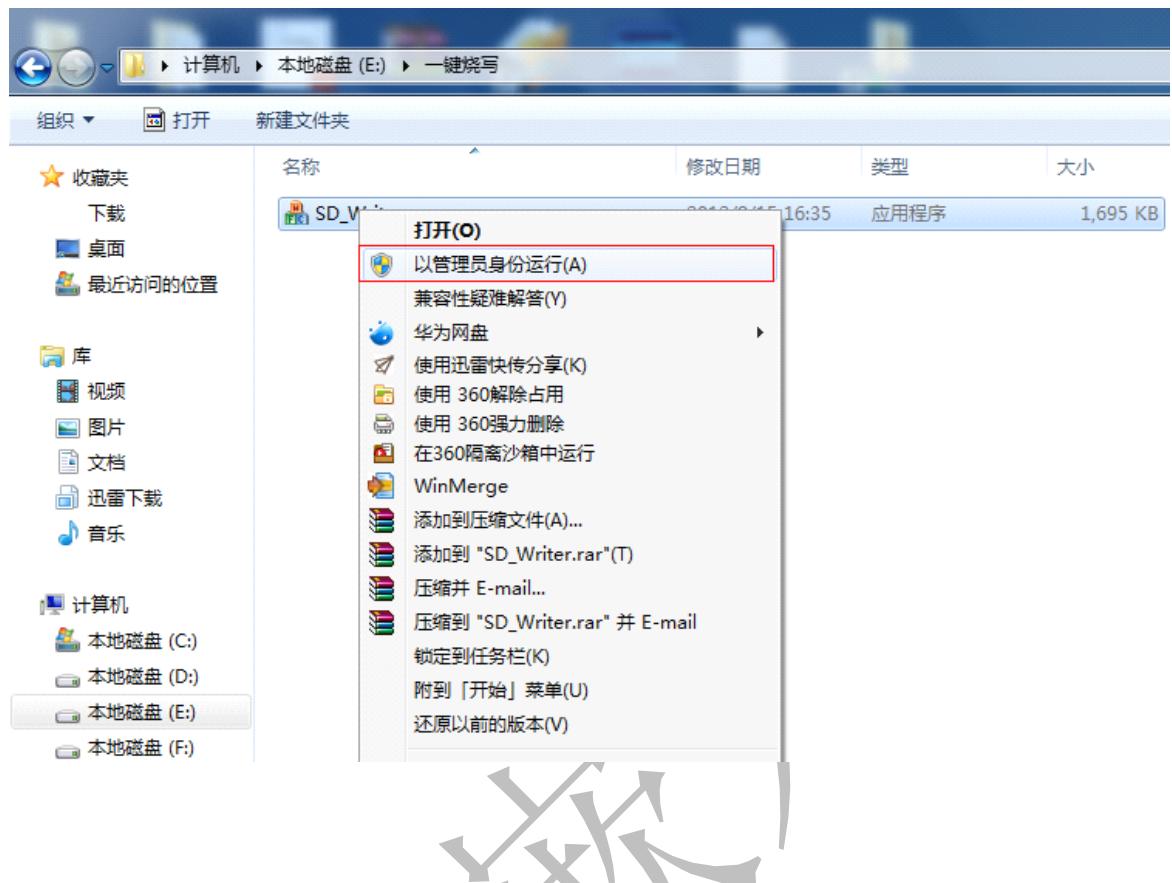


步骤 2. 通过 **SD_Writer.exe** 将 **mmc.bin** 烧写到 **SD** 卡中。
打开 **SD_Writer.exe**，下图是 xp 系统中 **SD_Writer.exe** 界面截图



注意: **Board Model** 选择 **6410**。**OSType** 选择 **Linux || Android** 选项。

下图是 WIN7 系统中 SD_Writer.exe 界面截图。注意 **Window7** 系统上运行烧写软件时
需要以管理员身份运行, 如图:



步骤 3. 点击“Scan”，这个步骤是自动搜寻 SD 卡所在盘符。

如果“Scan”没有正确设置 SD 卡所在盘符，就需要手动调整 SD Volume，把盘符调整为 SD 卡所在盘符（比如说，PC 的 USB 口接了两个或者两个以上的 U 盘或者 SD 卡，就有可能错误到扫描 SD 卡盘符）。

步骤 4. 将“SD Type”更改为 auto。这个步骤是为了让 SD_Writer 自动识别 SD 卡类型。

如果您的 PC 系统是 WIN7，您还需要点击“Format”来格式化 SD 卡。XP 用户看不到“Formart”，也不需要“Format”。这一点，是 XP 和 WIN7 用户操作中唯一的区别。

步骤 5. 将“OS Type”更改为 Linux。这个步骤是选择要烧写的系统类型。



步骤 6. 点击 “Select Boot”，选择适合自己开发板的 mmc.bin

mmc_ram128.bin 适用于 128M 内存的开发板

mmc_ram256.bin 适用于 256M 内存的开发板



步骤 7. 点击“Program”，出现“It's OK”表示操作成功。成功后如下图。



步骤 8. 点击“确定”，然后点击“Quite”。退出 SD_Writer.exe。

步骤 9. 首先，将 u-boot.bin 拷贝到 SD 卡中。

u-boot_ram128.bin 专门用于 **128M** 内存开发板。

u-boot_ram256.bin 专门用于 **256M** 内存开发板。

将与开发板对应的 u-boot 拷贝到 SD 卡中。接着在 SD 卡中将文件名改为 u-boot.bin 即可。

然后，将 zImage 拷贝到 SD 卡中。zImage 是 Linux 的内核映像文件。

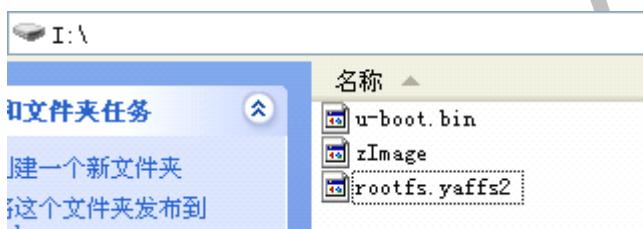
最后，将 rootfs.yaffs2 拷贝到 SD 卡中。

rootfs.yaffs2-nand256m 专门用于 **128M** 内存，**256M NandFlash** 开发板。

rootfs.yaffs2-nand2g 专门用于 **256M** 内存，**1G 或 2G 或者 4G Nandflash** 的开发板。

将与开发板对应的 yaffs2 文件拷贝到 SD 卡中。接着在 SD 卡中将文件名改为 rootfs.yaffs2 即可。

经过上述步骤操作后，正确的文件和文件名如图所示：



注意：PC 系统中，有一项功能是“隐藏已知文件名后缀”，如果在 PC 中开启这个功能，u-boot.bin 可能只能看到 u-boot 文件名。编写本文时，编写者已经将编写本文的电脑的“隐藏已知文件名后缀”功能关闭。

2-3 烧写Linux到开发板的NandFlash中

步骤 1. 将制作好的 SD 卡插入开发板 SD 的插槽。如图：



步骤 2. 接好 5V 直流电源（飞凌提供此电源，请使用飞凌提供的电源）。
开发板电源连接如图：



步骤 3. 拨码开关设置为 SD 卡启动。

拨码开关在底板 SD 卡启动的拨码开关设置如下：

引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
SD 卡启动	1	1	1	1	1	0	0	0

注：上表中。1 表示拨码需要调整到 On；0 表示拨码需要调整到 Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。

拨码开关设置 SD 卡启动如图所示：



步骤 4. 将飞凌提供的串口延长线连接开发板的 COM0 和 PC 机的串口（飞凌提供的串口线仅限于开发板和 PC 的连接，连接其他设备请先确定串口的线序）。

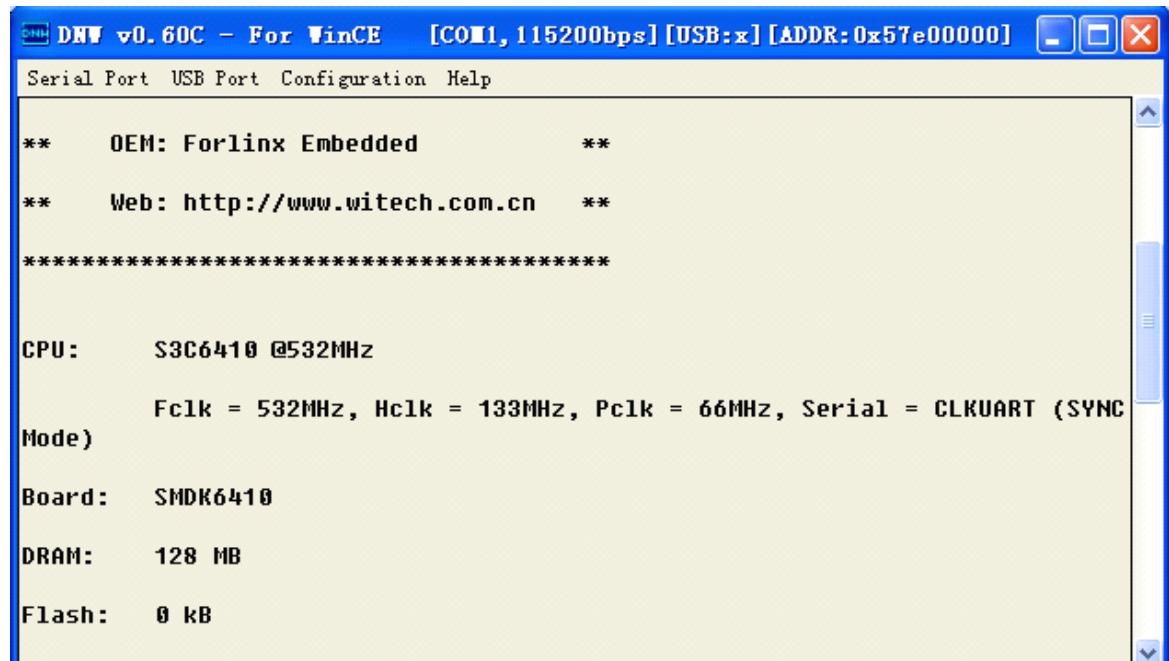


步骤 5. 打开飞凌提供的DNW软件，设置好DNW的串口（附录:dnw软件的使用简便教
-22-

程)

步骤 6. 拨动电源开关, 给开发板上电。会出现如下的串口信息, 因为信息较多, 贴出开始时和结束后的串口信息:

开始时:



```

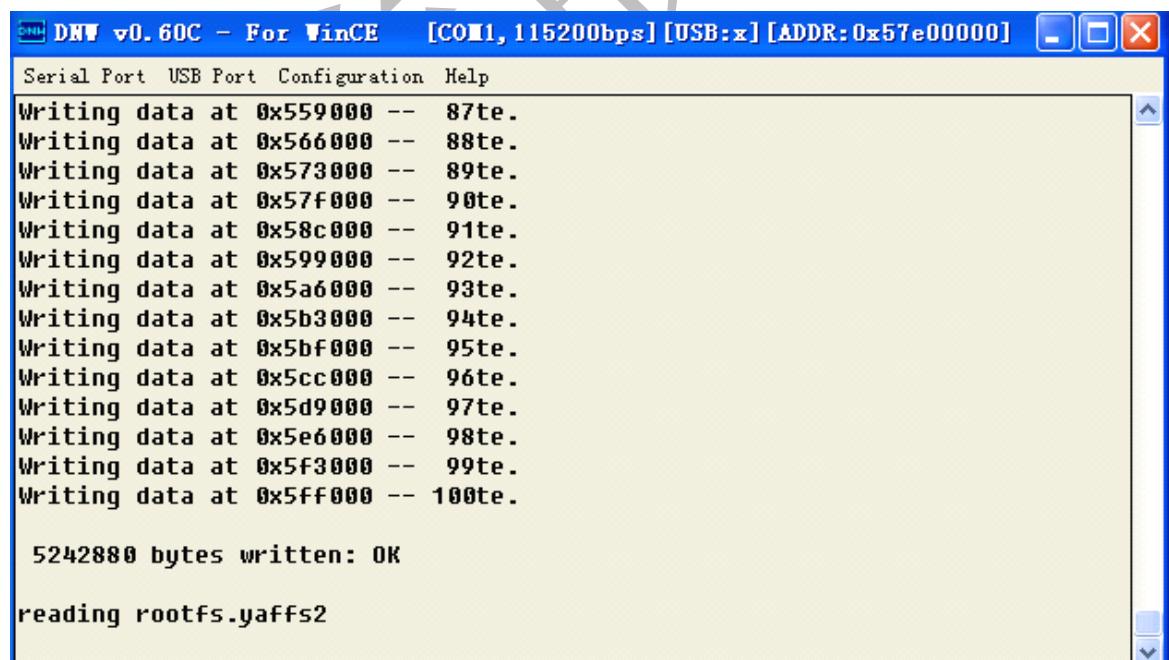
DNW v0.60C - For WinCE [COM1, 115200bps] [USB:x] [ADDR:0x57e00000] - X
Serial Port USB Port Configuration Help

** OEM: Forlinx Embedded **
** Web: http://www.witech.com.cn **

*****
CPU: S3C6410 @532MHz
      Fclk = 532MHz, Hclk = 133MHz, Pclk = 66MHz, Serial = CLKUART (SYNC Mode)
Board: SMDK6410
DRAM: 128 MB
Flash: 0 kB

```

烧写过程中:



```

DNW v0.60C - For WinCE [COM1, 115200bps] [USB:x] [ADDR:0x57e00000] - X
Serial Port USB Port Configuration Help

Writing data at 0x559000 -- 87te.
Writing data at 0x566000 -- 88te.
Writing data at 0x573000 -- 89te.
Writing data at 0x57f000 -- 90te.
Writing data at 0x58c000 -- 91te.
Writing data at 0x599000 -- 92te.
Writing data at 0x5a6000 -- 93te.
Writing data at 0x5b3000 -- 94te.
Writing data at 0x5bf000 -- 95te.
Writing data at 0x5cc000 -- 96te.
Writing data at 0x5d9000 -- 97te.
Writing data at 0x5e6000 -- 98te.
Writing data at 0x5f3000 -- 99te.
Writing data at 0x5ff000 -- 100te.

5242880 bytes written: OK

reading rootfs.yaffs2

```

注意: 烧写完成后, 蜂鸣器会发出滴滴的提示声, 如果没有提示说明烧写不成功, 另外烧写的进度会在 4.3 寸 LCD 上显示, 如果您使用的是 4.3 寸 LCD, 完全不需要 DNW 显示烧写信息, 可以实现使用 SD 卡批量烧写 Linux 系统, 非常适合于产品型客户。

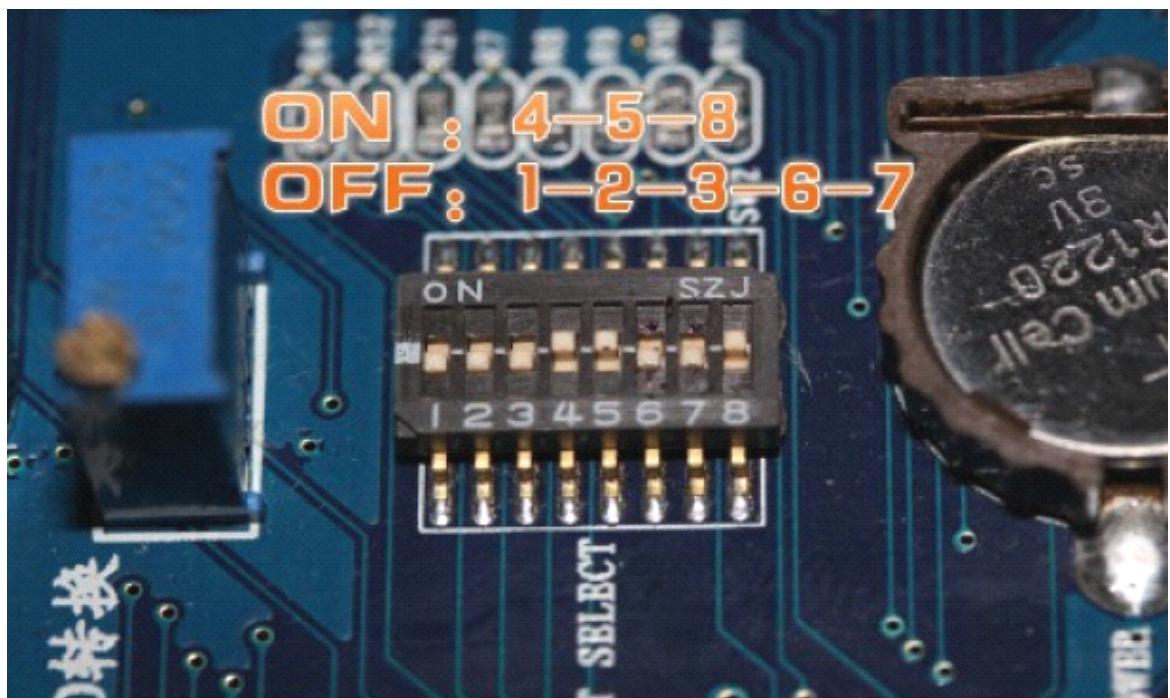
步骤 7. 拨动电源开关，开发板断电，将拨码开关设置为 nand flash 启动。设置如下

引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
Nand 卡启动	1	0	0	1	1	0	0	0

注：上表中。1 表示拨码需要调整到 On；0 表示拨码需要调整到 Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。

拨码开关设置 NandFlash 启动如图所示：



步骤 8. 重新开启电源，Linux 系统可以正常启动了。

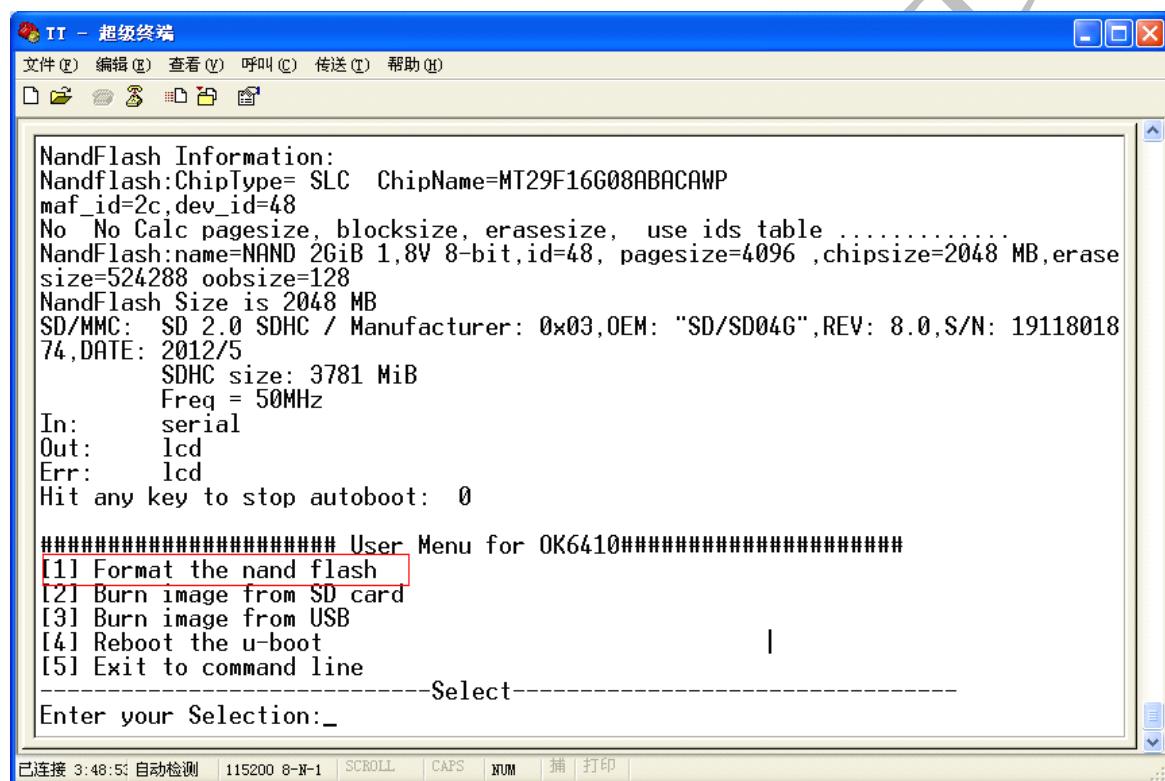
2-4 出现坏块怎么办？

对于坏块的处理：使用 NandFlash，免不了出现个别的坏块。没有关系，一般的坏块就交给飞凌 6410 开发板自己处理吧。飞凌 6410 开发板对坏块有一套完善的处理机制。（源码详见 NAND 驱动）。

如果出现因为坏块无法启动 Linux 操作系统，那就需要一个方法来处理这些逻辑上的坏块（实际上坏块不一定是真的坏了）。逻辑坏块引起的系统无法启动，可以使用下面这种方法：

步骤 1. 先用串口线连接好开发板 COM0 与 PC 机的串口，打开并设置 DNW 软件（附录:dnw软件的使用简便教程）。

步骤 2. 然后给开发板上电，等到出现延时 1 秒启动系统时，在 DNW 软件中按 PC 键盘的空格键使开发板停留在 uboot 状态。因为停留时间只有 1 秒，所以需要很快的按下空格键。如图：



输入 1，代表擦除整个 NandFlash，包括擦除伪坏块。



```
TT - 超级终端
文件(E) 编辑(B) 查看(V) 呼叫(C) 传送(T) 帮助(H)
□ ☐ ×

SDHC size: 3781 MiB
Freq = 50MHz
In: serial
Out: lcd
Err: lcd
Hit any key to stop autoboot: 0

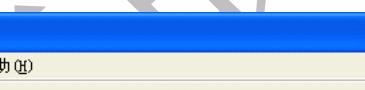
#####
User Menu for OK6410#####
[1] Format the nand flash
[2] Burn image from SD card
[3] Burn image from USB
[4] Reboot the u-boot
[5] Exit to command line
-----Select-----
Enter your Selection:1

NAND scrub: device 0 whole chip
Warning: scrub option will erase all factory set bad blocks!
        There is no reliable way to recover them.
        Use this command only for testing purposes if you
        are sure of what you are doing!

Really scrub this NAND flash? <y/N>
-
```

已连接 3:51:55 自动检测 | 115200 8-N-1 | SCROLL | CAPS | NUM | 捕 | 打印 |

输入 y,回车，即可擦除 NandFlash。



```
TT - 超级终端
文件(E) 编辑(B) 查看(V) 呼叫(C) 传送(T) 帮助(H)
□ ☐ ×

-----Select-----
Enter your Selection:1

NAND scrub: device 0 whole chip
Warning: scrub option will erase all factory set bad blocks!
        There is no reliable way to recover them.
        Use this command only for testing purposes if you
        are sure of what you are doing!

Really scrub this NAND flash? <y/N>
Erasing at 0x5c280000 -- 72% complete.
NAND 4GiB 1.8V 8-bit: MTD Erase failure: -5
Erasing at 0x7ff80000 -- 100% complete.
Scanning device for bad blocks
OK

#####
User Menu for OK6410#####
[1] Format the nand flash
[2] Burn image from SD card
[3] Burn image from USB
[4] Reboot the u-boot
[5] Exit to command line
-----Select-----
Enter your Selection:
```

已连接 3:54:06 自动检测 | 115200 8-N-1 | SCROLL | CAPS | NUM | 捕 | 打印 |

这时候，整个开发板的 NandFlash 会被清空，坏块也会被处理。您可以使用一键烧写 Linux 的方法把 linux 重新烧写一遍。

注意：在开发和学习的过程免不了产生一些逻辑上的坏块，用上面的方法可以处理。但是 nand scrub 命令本身不宜经常使用，可能会对 nand 进行错误的校正。这些坏块是 nand 的一个特征，如果您有兴趣，可以进一步了解 nand 的特性和飞凌 6410 对 nand 的处理机制。



第三章 USB 烧写 Linux

本章详细介绍了烧写 Linux 映像到开发板的方法。该部分内容在 Windows XP SP3 上测试通过。在本章中，您将学会使用 Uboot 菜单选项来快速烧写 Linux 系统。

本章所需文件路径：

文件名（文件用途）	文件在基础光盘中的路径
SD_Writer.exe(PC 烧写工具)	Linux-3.0.1\Linux 烧写工具\
mmc_ram128.bin (128M 内存开发板适用的 sdboot, 用于 sd 卡启动)	Linux-3.0.1\Linux 烧写工具\
mmc_ram256.bin (256M 内存开发板适用的 sdboot, 用于 sd 卡启动)	Linux-3.0.1\Linux 烧写工具\
u-boot_ram128.bin (uboot 映像, 适用于 128M 内存开发板)	Linux-3.0.1\demo\
u-boot_ram256.bin (uboot 映像, 适用于 256M 内存开发板)	Linux-3.0.1\demo\
zImage(内核映像)	Linux-3.0.1\demo\
rootfs.yaffs2-nand256m (用于触摸屏输入的 yaffs2 文件系统, 适用于 256Mnandflash 的开发板)	Linux-3.0.1\filesystem\
rootfs.yaffs2-nand2g (用于触摸屏输入的 yaffs2 文件系统, 适用于 1G 字节或 2G 字节或者 4G 字节的 nandflash 的开发板)	Linux-3.0.1\filesystem\
XP 系统的 DNW 的 USB 驱动	实用工具\USB 驱动\DNW 下载驱动
DNW(串口、USB 口工具)	实用工具\

注意：如果您的开发板安装的是 **WinCE** 系统，现在要安装 **Linux** 系统了，请参考第二章中 **2-1 对于 WinCE 系统换 Linux 系统的特殊说明** 一节擦除 **NandFlash**。

3-1 制作用于USB烧写Linux的SD卡

步骤 1: 将 SD 卡格式化为 FAT32 格式

将 SD 卡接入 SD 读卡器中，把 SD 读卡器插在 PC 机的 USB 口中。



等到 PC 机能够正常识别出 SD 卡后，把 SD 卡格式化为 FAT32 格式。



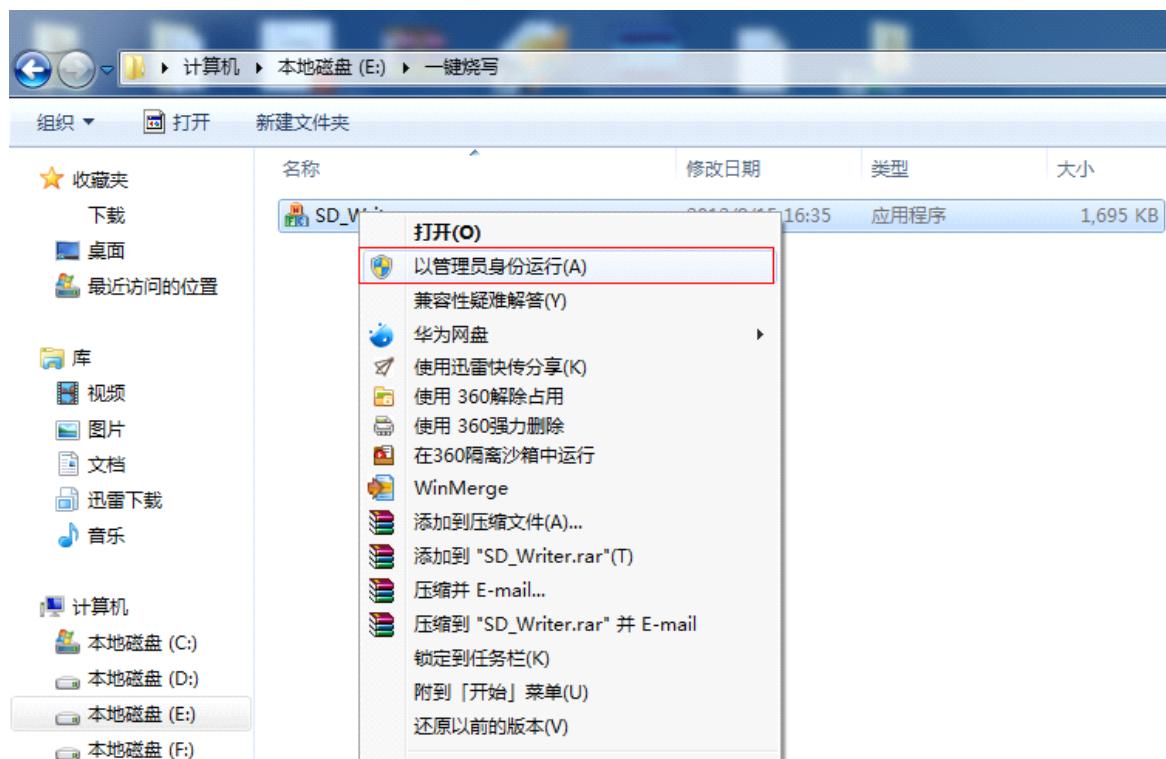
步骤 2. 通过 SD_Writer.exe 将 mmc.bin 烧写到 SD 卡中
打开 SD_Writer.exe。

下图是 xp 系统中 SD_Writer.exe 界面截图

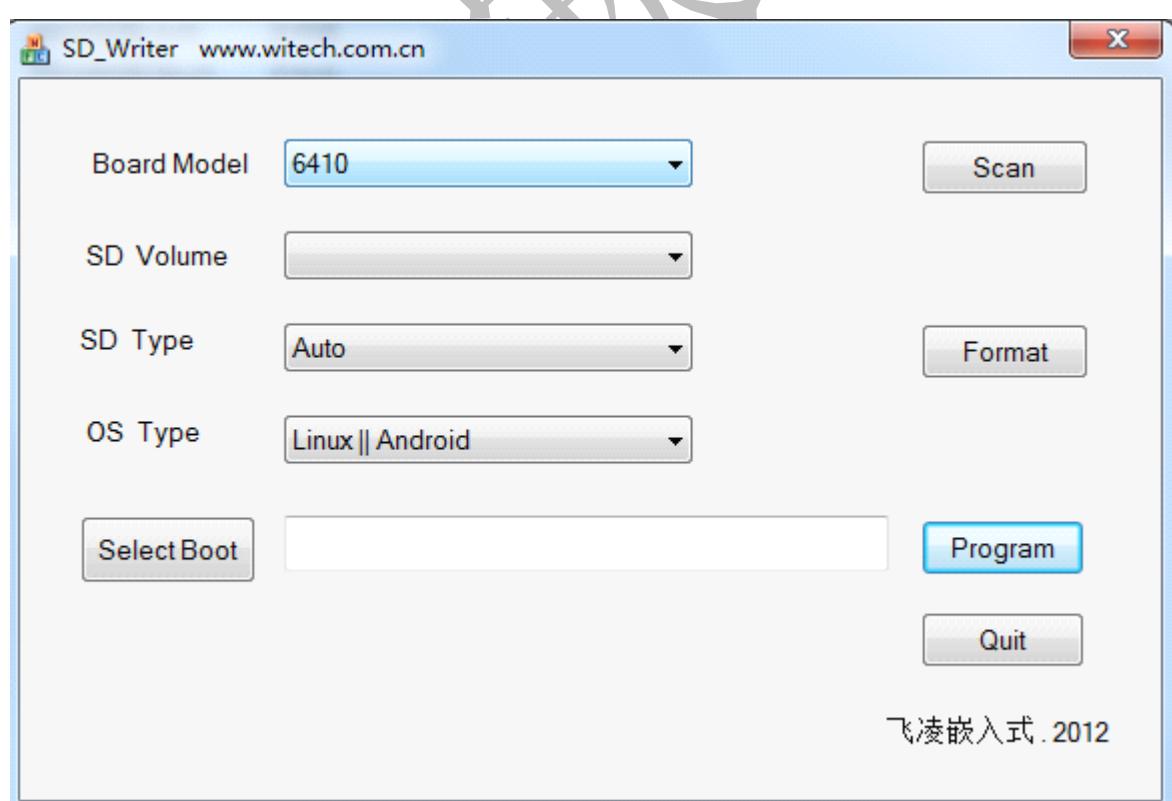


下图是 WIN7 系统中 SD_Writer.exe 界面截图，在 Win7 下面运行 SD_Writer 需要使用管理员权限，否则无法烧写 mmc.bin 文件到 SD 卡中。





启动以后如下图所示：



步骤 3. 点击“Scan”，这个步骤是自动搜寻 SD 卡所在盘符。

步骤 4. 将“SD Type”更改为 auto。这个步骤是为了让 SD_Writer 自动识别 SD 卡类型。

如果您的 PC 是 WIN7，您还需要点击“Format”来格式化 SD 卡。XP 用户看不到“Format”，也不需要“Format”。这一点，是 XP 和 WIN7 用户操作中唯一的区别。

步骤 5. 将“OS Type”更改为 Linux。这个步骤是选择要烧写的系统类型。

步骤 6. 点击 “Select Boot”，选择适合自己开发板的 mmc.bin

mmc_ram128.bin 适用于 128M 内存的开发板

mmc_ram256.bin 适用于 256M 内存的开发板



步骤 7. 点击“Program”，出现“It's OK”表示操作成功。成功后如下图。



注意：此时在 PC 机上查看 SD 卡时，SD 卡上没有任何显示文件。

3-2 USB烧写Linux到开发板的NandFlash

3-2-1 设置开发板从SD卡启动

步骤 1. 将制作好的 SD 卡插入开发板 SD 的插槽。如图：



步骤 2. 接好 5V 直流电源（飞凌提供此电源，请使用飞凌提供的电源）。

开发板电源连接如图：



步骤 3. 拨码开关设置为 SD 卡启动。

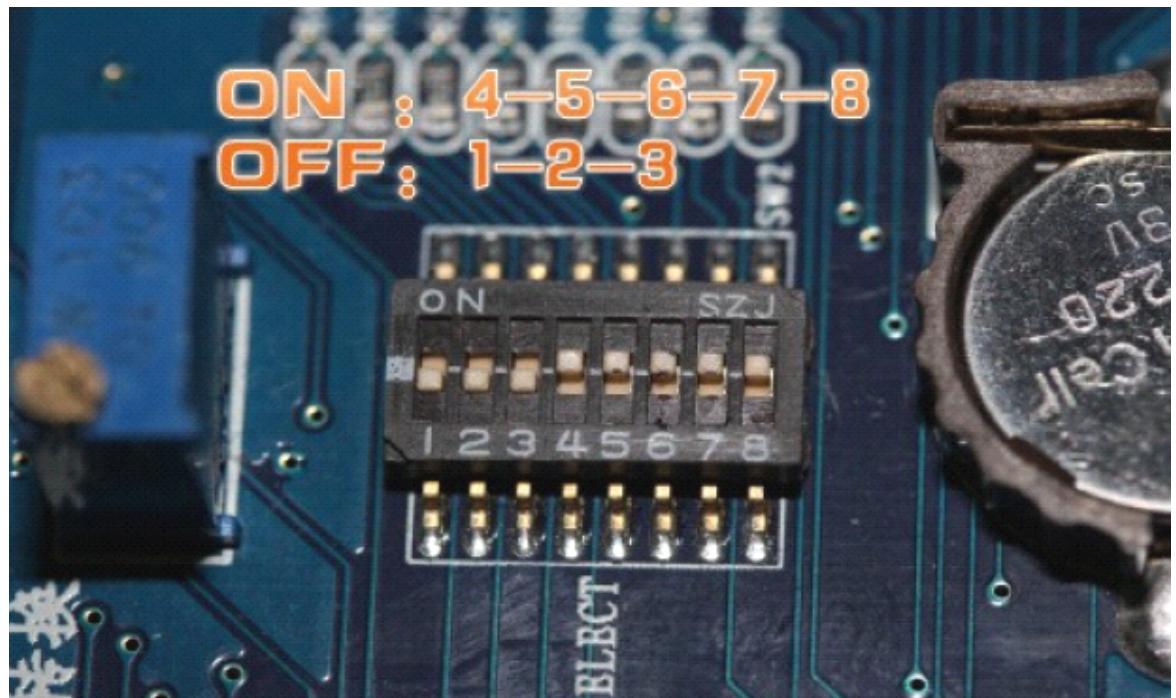
拨码开关在底板 SD 卡启动的拨码开关设置如下：

引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
SD 卡启动	1	1	1	1	1	0	0	0

注：上表中。1 表示拨码需要调整到 On；0 表示拨码需要调整到 Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。

拨码开关设置 SD 卡启动如图所示：

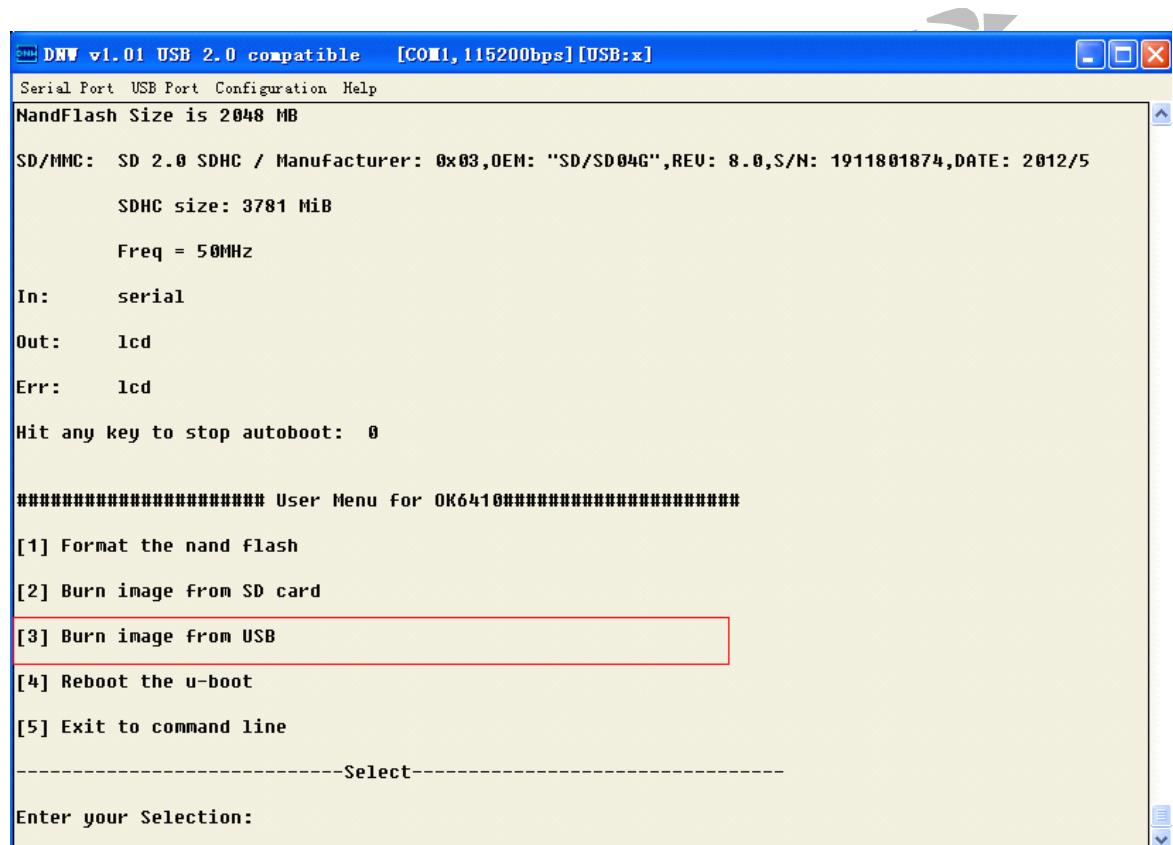


步骤 4. 将飞凌提供的串口延长线连接开发板的 COM0 和 PC 机的串口（飞凌提供的串口线仅限于开发板和 PC 的连接，连接其他设备请先确定串口的线序）。



步骤 5. 打开飞凌提供的DNW软件，设置好DNW的串口（附录:dnw软件的使用简便教程）

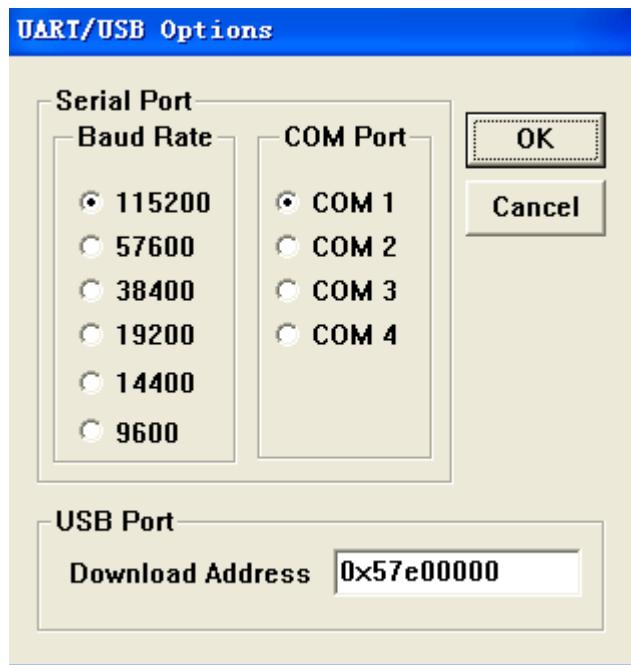
步骤 6. 然后给开发板上电，等到出现延时 1 秒启动系统时，在 DNW 软件中按 PC 键盘的空格键使开发板停留在 **uboot** 状态。因为停留时间只有 1 秒，所以需要很快的按下空格键。



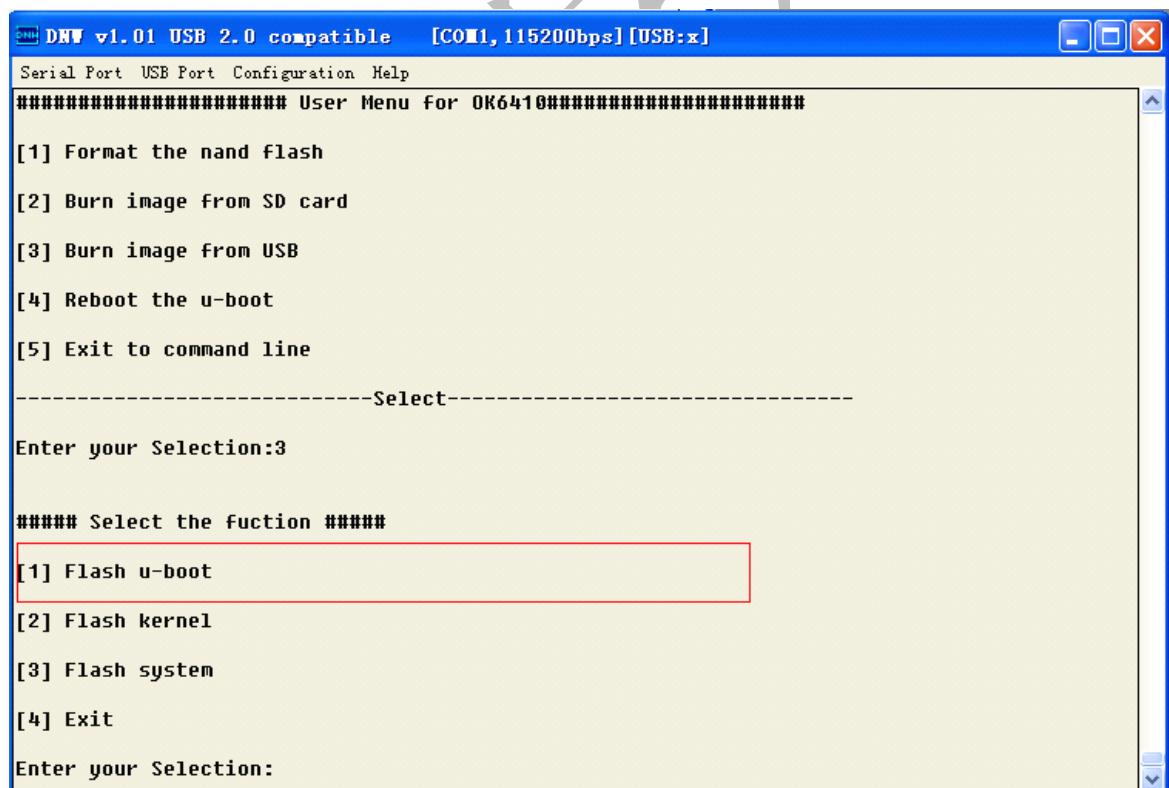
输入 3，就开始使用 USB 下载 Uboot，内核，文件系统了。

3-2-2 下载并烧写u-boot到Nandflash

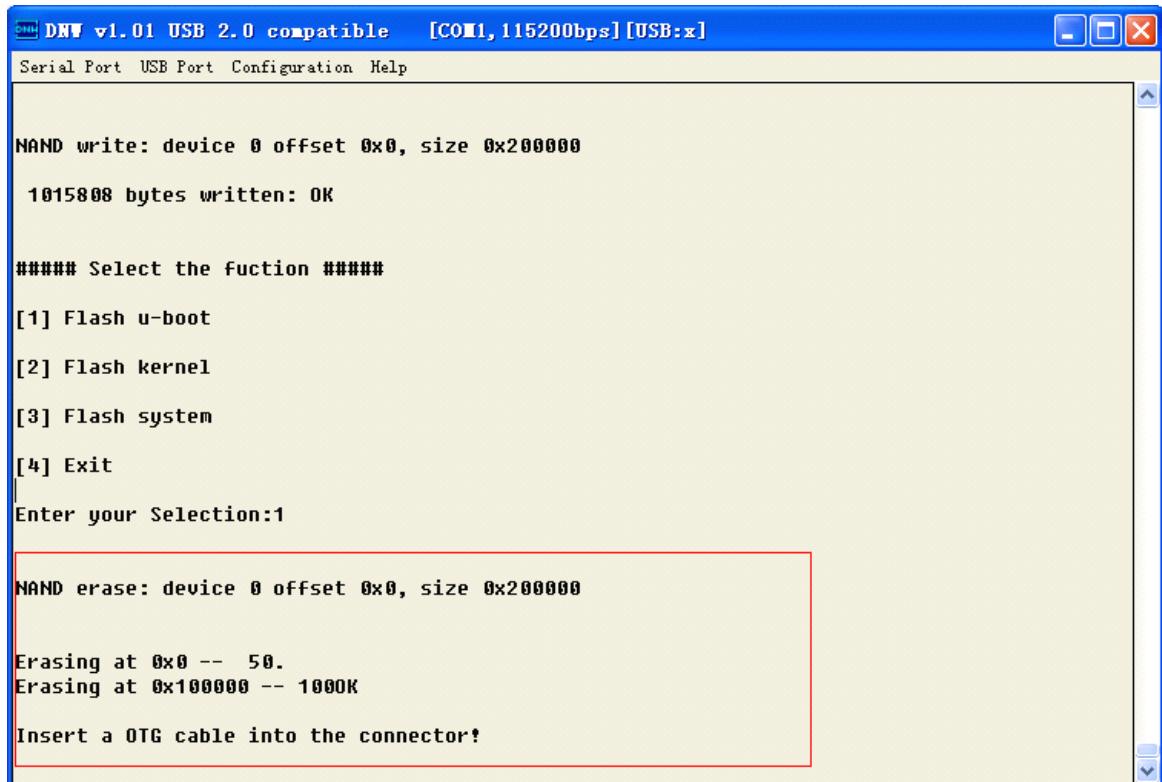
步骤 1. 点击 DNW 菜单 ‘Configuration-->Options’ 弹出 DNW 配置窗口，将 USB Port 的 Download Address 值设置为 0x57e00000



步骤 2. 启动 Uboot，按空格键，出现菜单后，选择 3，然后出现下面的菜单



输入 1，开始下载 Uboot，如果您的电脑上未安装 USB DNW 驱动，这时您的电脑会提示发现新设备，这时需要按照驱动了，如何安装驱动，请参考 [3-3 安装 USB 下载专用驱动](#)



```
DNW v1.01 USB 2.0 compatible [COM1, 115200bps] [USB:x]
Serial Port USB Port Configuration Help

NAND write: device 0 offset 0x0, size 0x2000000
1015808 bytes written: OK

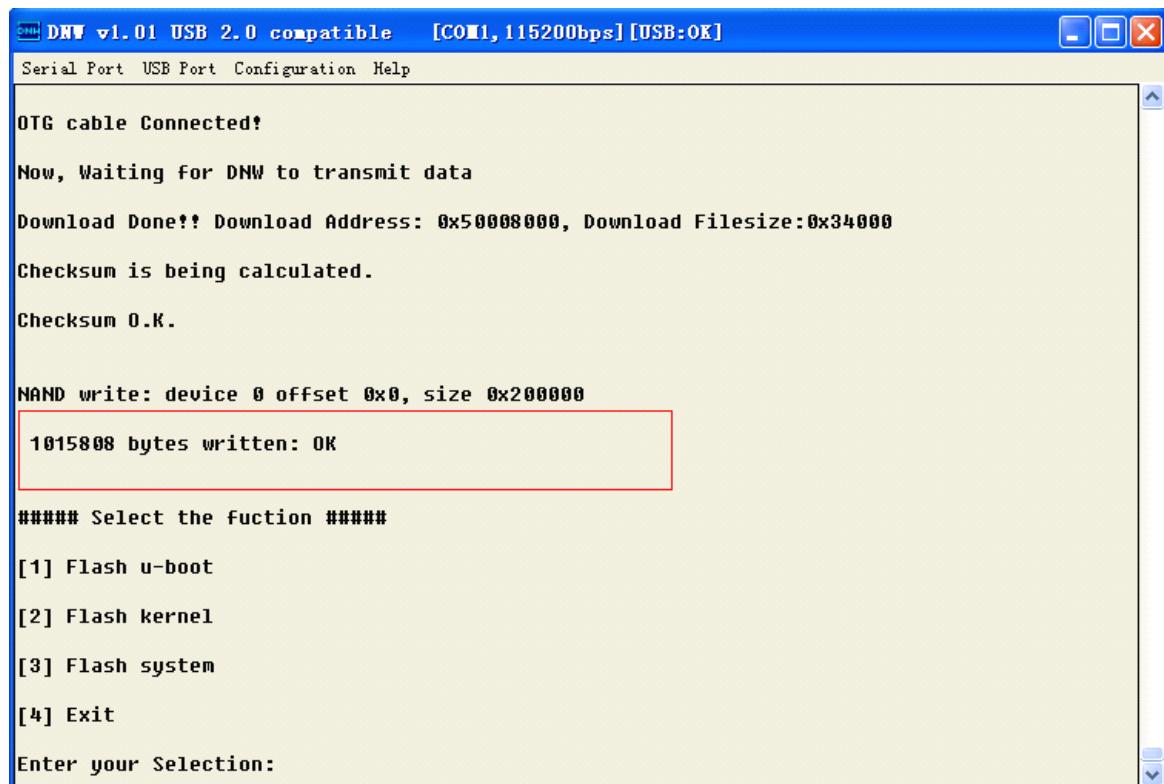
##### Select the fuction #####
[1] Flash u-boot
[2] Flash kernel
[3] Flash system
[4] Exit
Enter your Selection:1

NAND erase: device 0 offset 0x0, size 0x2000000
Erasing at 0x0 -- 50.
Erasing at 0x100000 -- 1000K
Insert a OTG cable into the connector!
```

把 USB 线的一端插入到开发板的 OTG 口，执行下面的步骤。

步骤 3. 将 U-boot 映像文件下载到内存：点击 DNW 菜单 “USB Port-->Transmit-->Transmit” 在弹出的文件浏览窗口中选择 u-boot , u-boot_ram128.bin 专门用于 128M 内存开发板。 u-boot_ram256.bin 专门用于 256M 内存开发板。





The screenshot shows the DNW v1.01 software interface. The title bar reads "DNW v1.01 USB 2.0 compatible [COM1, 115200bps] [USB:OK]". The menu bar includes "Serial Port", "USB Port", "Configuration", and "Help". The main window displays the following text:

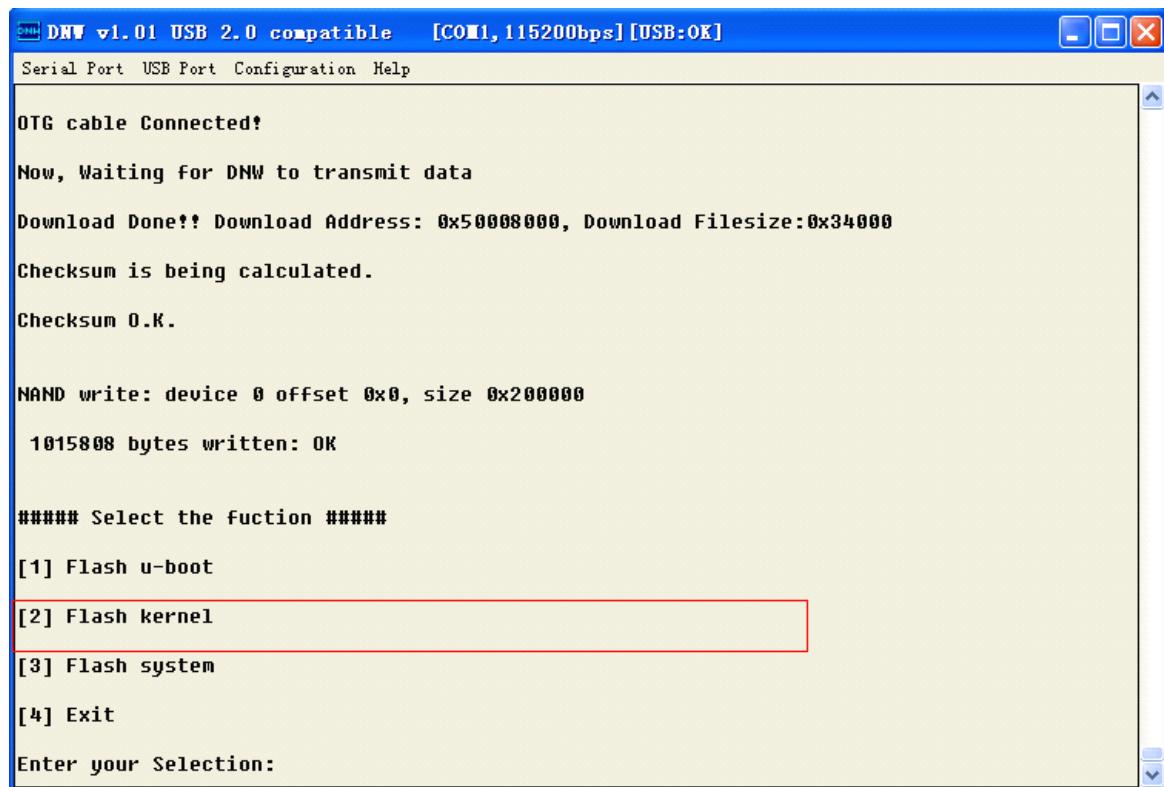
```
OTG cable Connected!
Now, Waiting for DNW to transmit data
Download Done!! Download Address: 0x50008000, Download Filesize:0x34000
Checksum is being calculated.
Checksum O.K.

NAND write: device 0 offset 0x0, size 0x200000
1015808 bytes written: OK
##### Select the fuction #####
[1] Flash u-boot
[2] Flash kernel
[3] Flash system
[4] Exit
Enter your Selection:
```

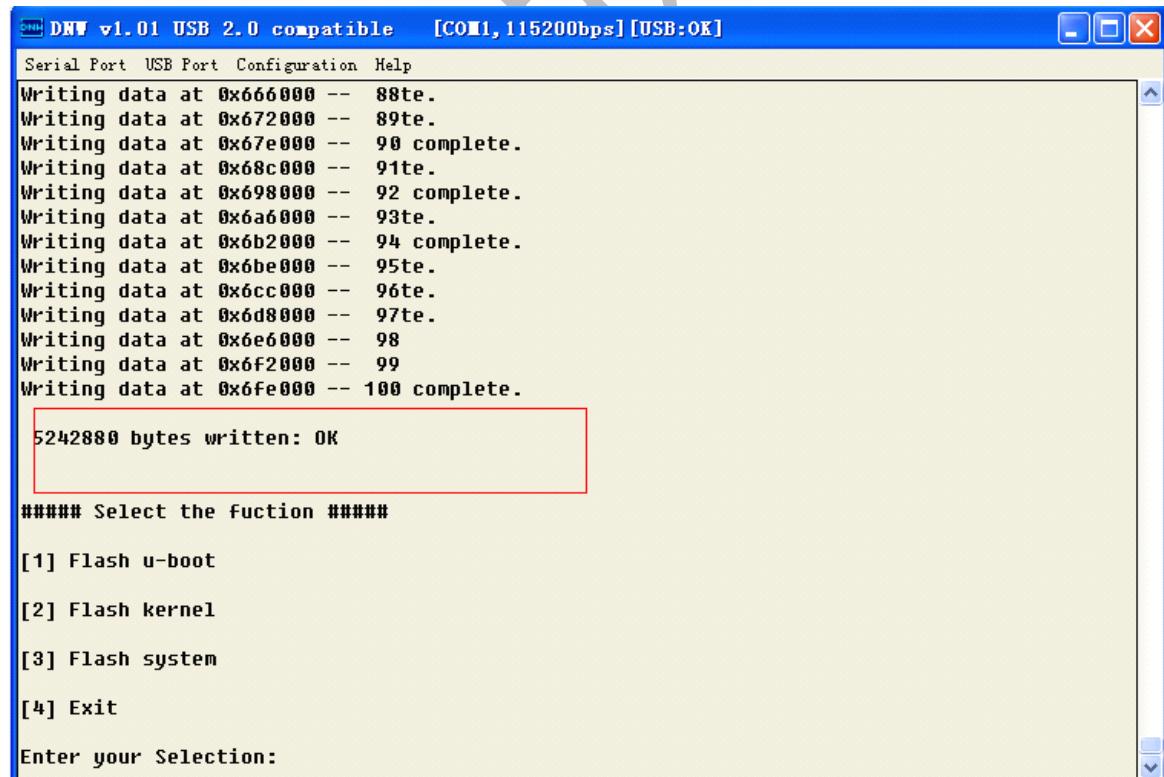
注意：如果点击“USB Port-->Transmit-->Transmit”后出现“Can't open USB device”，可以点击“确定”后，多尝试几次。这个问题可能是由于三星提供的USB驱动所致。

3-2-3 烧写Kernel

烧写完成 Uboot 后，执行命令 2, 开始下载内核文件。



点击 DNW 菜单 “USB Port-->Transmit-->Transmit” 在弹出的文件浏览窗口中选择 zImage 文件，这样就开始烧写 Linux 内核文件了。

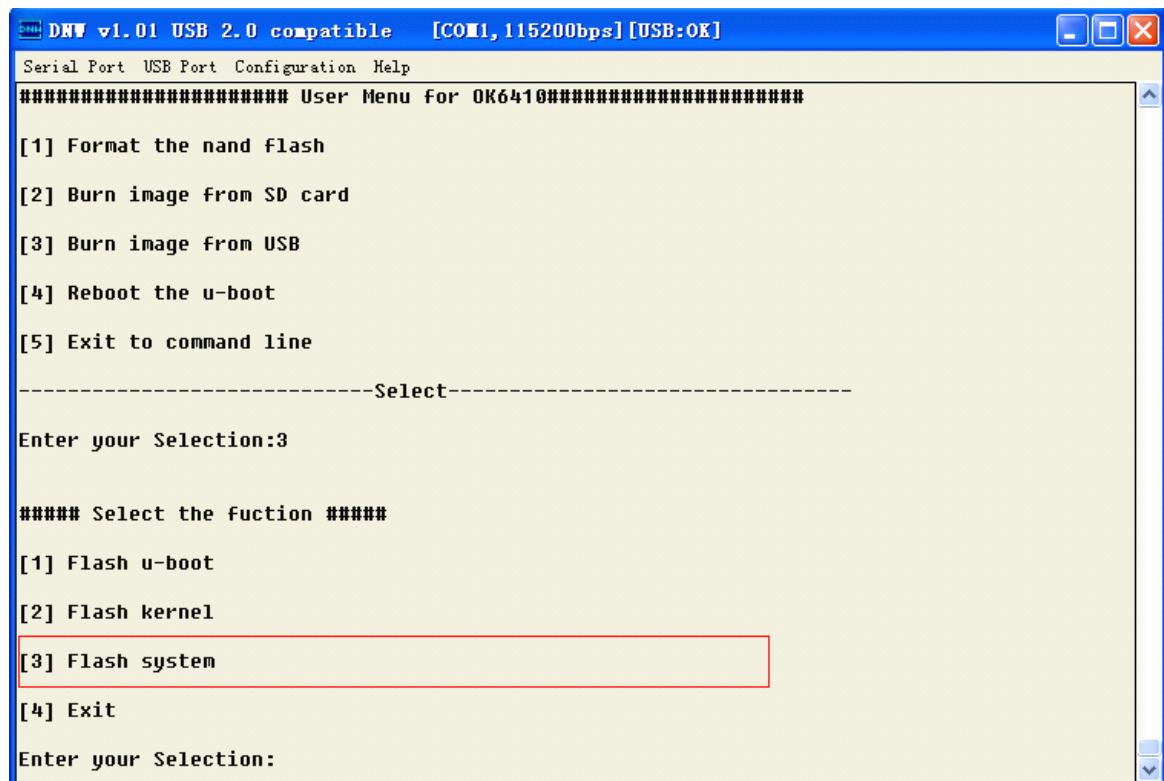


3-2-4 烧写根文件系统

目前飞凌 6410 Linux 支持 yaffs2 文件系统。

yaffs 是第一个专门为Nandflash存储器设计的嵌入式文件系统,适用于大容量的存储设备;并且是在 GPL (General Public License) 协议下发布的,可在其网站免费获得源代码。

步骤 1. 进入 U-boot 命令菜单以后, 输入 3 进入 USB 下载界面。再次输入 3 选择更新文件系统, 如图:



点击 DNW 菜单 “USBPort-->Transmit-->Transmit” 在弹出的文件浏览窗口中选择 yaffs2 文件, 开始烧写文件系统到 NandFlash。

rootfs.yaffs2-nand256m 专门用于 256MNandflash 开发板。

rootfs.yaffs2-nand2g 专门用于 1G 字节或 2G 字节或者 4G 字节的 Nandflash 开发板。

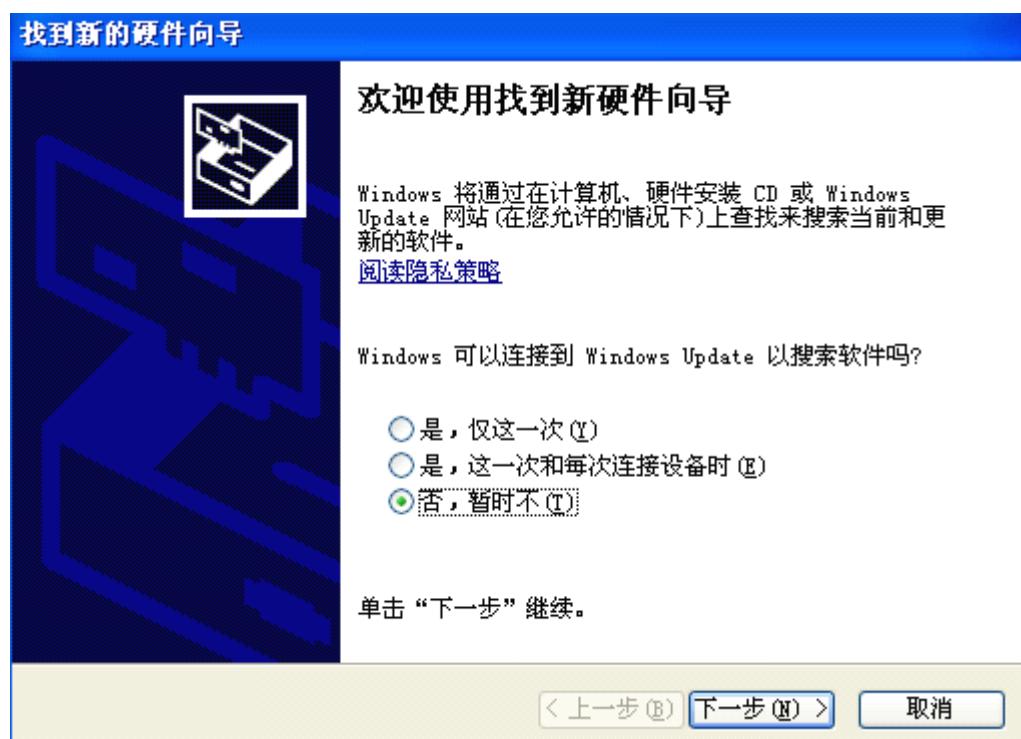
注意: 文件系统占用空间比较大, 烧写过程需要一定的时间, 烧写过程中请认真的等待, 我们的文件系统中有两个版本的 **Qt**, **Qtopia2.2.0** 和 **Qt/E4.7**, 可以根据您的需求, 保留或者删除 **Qt** 库, 这样会大大减少文件系统的空间。

3-3 安装USB下载专用驱动

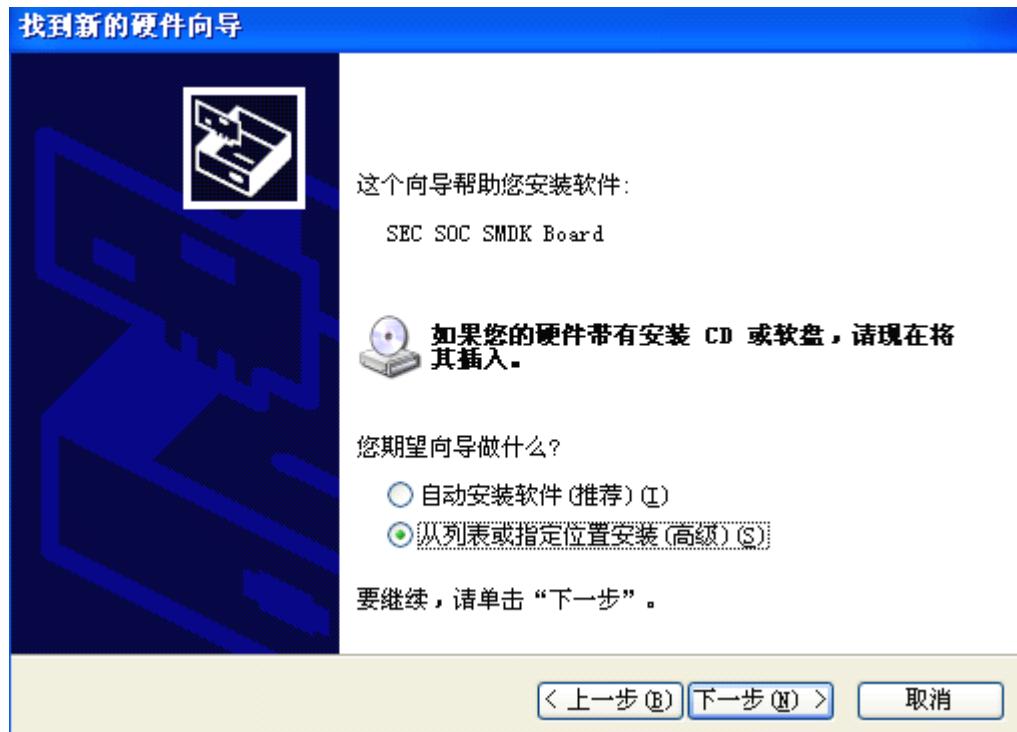
特殊说明: **USB** 下载驱动专用于 **DNW** **USB** 下载功能。开发板运行 **U-boot** 命令行状态下, 下载安装 **Uboot** 命令时, 方可安装 **USB** 下载驱动, 其他情况都不可以安装或使用下

载功能。

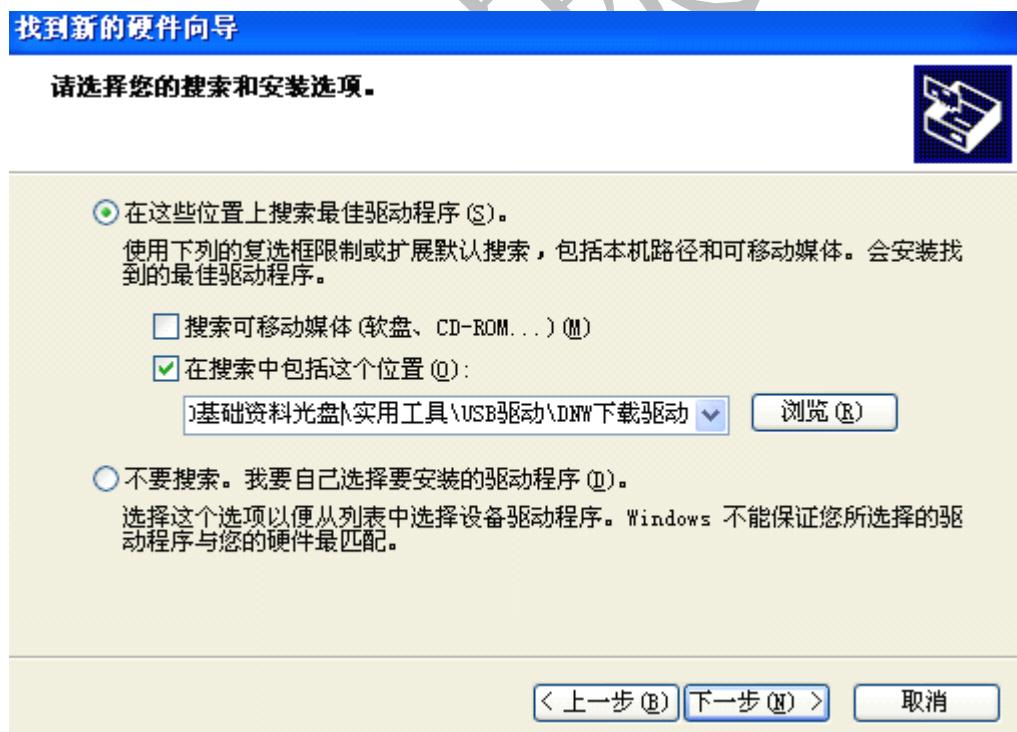
步骤 1. 第一次使用 USB 下载文件时，电脑会提示找到新硬件，并弹出“找到硬件向导”窗口。

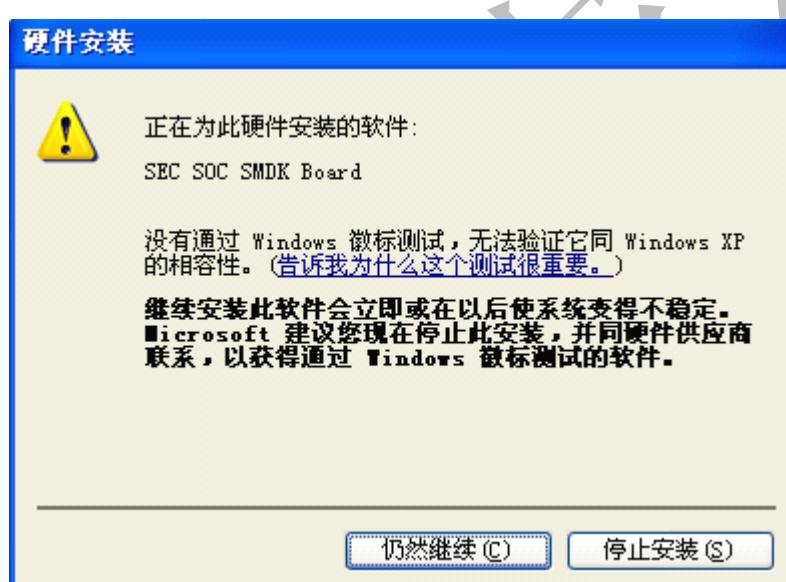


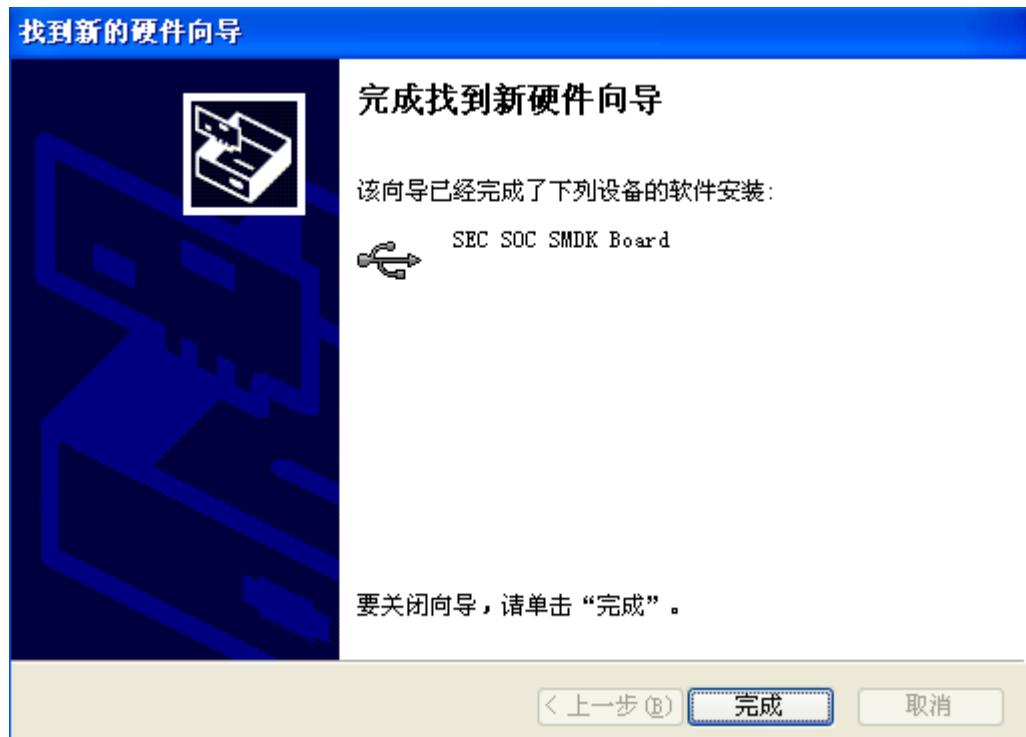
步骤 2. 这里我们选择从“列表或指定位置安装”点击下一步



步骤 3. 选中“在搜索中包括这个位置”，点击浏览按钮指定驱动所在位置；驱动位于用户基础资料光盘的“实用工具\USB 驱动\DNW 下载驱动”文件夹中。点击‘下一步’，在弹出的窗口中选择‘oemX.inf’（X 的值不定），然后继续点击‘下一步’开始安装驱动程序：







步骤 4. 安装完成后在 DNW 标题栏上会显示 [**USB:OK**], DNW 窗口中会打印如下图所示信息:

The screenshot shows the DNW v0.60C terminal window. The title bar reads 'DNW v0.60C - For WinCE [COM1, 115200bps] [USB:OK] [ADDR:0x57e00000]'. The window content is a log of the system boot process:

```
*** Warning - bad CRC or moving NAND, using default environment

In:      serial
Out:     serial
Err:     serial

Hit any key to stop autoboot:  0

SMDK6410 #
SMDK6410 # dnw 50008000
Insert a OTG cable into the connector!
OTG cable Connected!
Now, Waiting for DNW to transmit data
```

3-4 设置开发板从Nandflash启动

SW2 引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
Nand 卡启动	1	0	0	1	1	0	0	0

注：上表中。1 表示拨码需要调整到 On；0 表示拨码需要调整到 Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。

拨码开关设置 NandFlash 启动如图所示：



第四章 测试 Linux-3.0.1

本章主要介绍飞凌 6410 Linux-3.0.1 驱动及相关服务测试，建议使用超级终端做控制台。

本章中，如没有特殊说明，命令是在开发板控制终端执行。在每条命令开头加符号 ‘#’ 以表明命令的开始。

4-1 Qtopia2.2.0、QT-Extended-4.4.3、QT/E-4.7.1 切换

开发板 Linux 系统烧写好之后，默认启动 Qtopia2.2.0。在开发部终端里，切换各 QT 界面的方法：

1. 切换 Qtopia2.2.0:

```
#qtopia &
```

2. 切换 QT-Extended-4.4.3:

```
#qtopia4 &
```

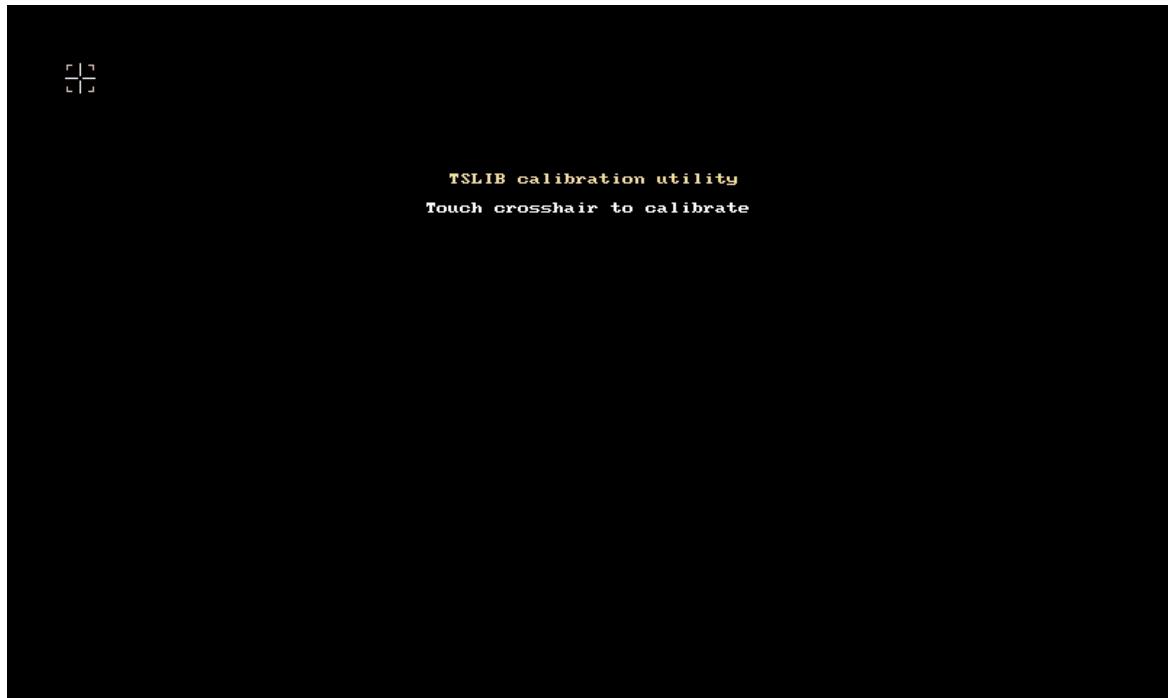
3. 切换 QT/E-4.7.1:

```
#qt4 &
```

注意：本次发布的文件系统 rootfs.yaffs2 里面默认是不含有 Qtopia4.4.3 版本，但是我们提供了编译好的压缩包 Qtopia4.4.3.tar.gz，位于光盘资料里面，您如果想要运行该版本，可以使用 SD 卡或者 U 盘把 Qtopia4.4.3.tar.gz 拷贝到开发板的 /opt 目录下面，解压，然后在超级终端或者 DNW 里面运行 qtopia4 & 即可。

4-2 触摸屏的校准与重新校准

每次重新烧写根文件系统之后，启动系统，会看到 tslib 的校准界面，如图：



校准方法：使用触摸笔依次点击屏幕中  的正中央。一共点击五次，每次触摸的点会不一样。五点点击完毕后，tslib 会在根文件系统/etc 中生成 pointercal 文件。Pointercal 是校准信息文件。

校准注意事项：在校准时，尽可能准确的使用触摸笔依次点击屏幕中  的正中央。如果随意点击五次也是可以通过校准，但是校准信息文件 pointercal 中的校准数据是错误的。

重新校准的方法：将该文件删除后重启开发板会自动进入 tslib 校准程序，命令如下：

`#rm /etc/pointercal` （删除校准文件）

`#reboot` （重新启动开发板）

注意：QTE是不能同时支持鼠标和触摸屏的，若用的是 支持鼠标的文件系统，则无法校准触摸。

本小节测试程序 tslib 源码位置：基础光盘\QTE 移植教程与源码\tslib.tar.gz

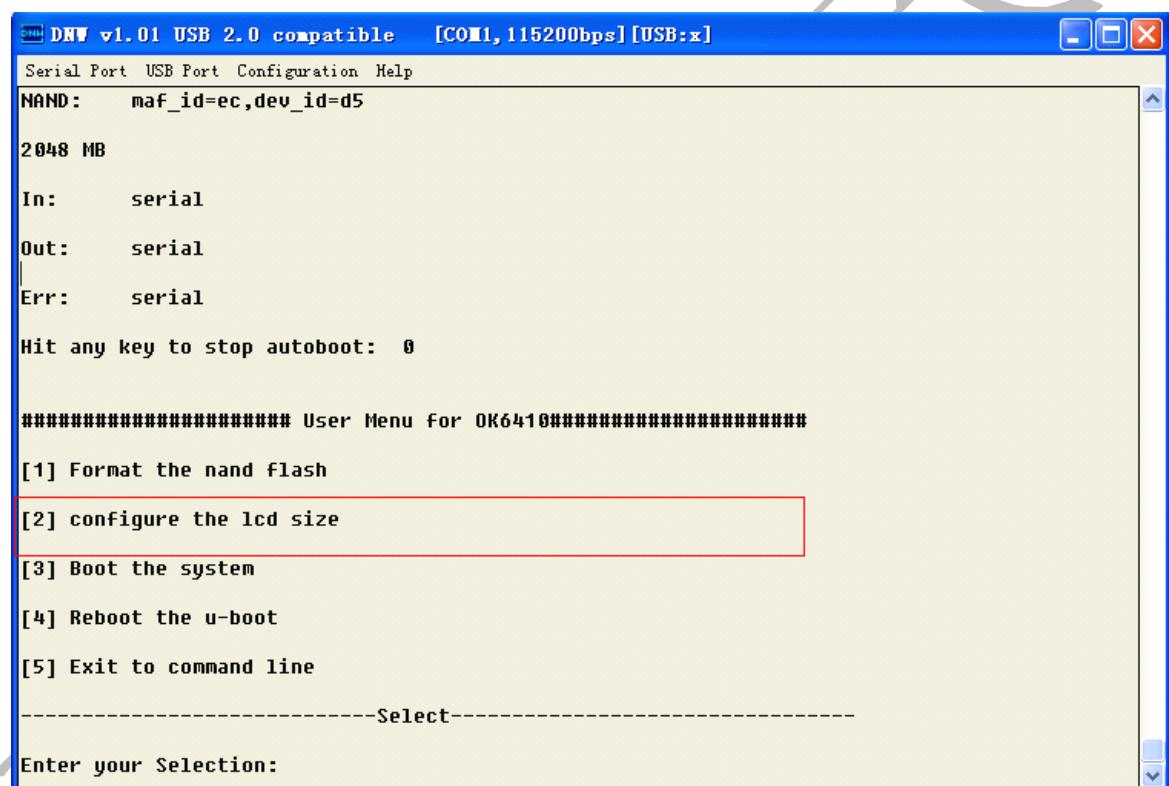
4-3 修改LCD分辨率

Linux-3.0.1是靠修改uboot参数来修改LCD分辨率的。修改分辨率，实质上是根据uboot参数，加载对应LCD的驱动。

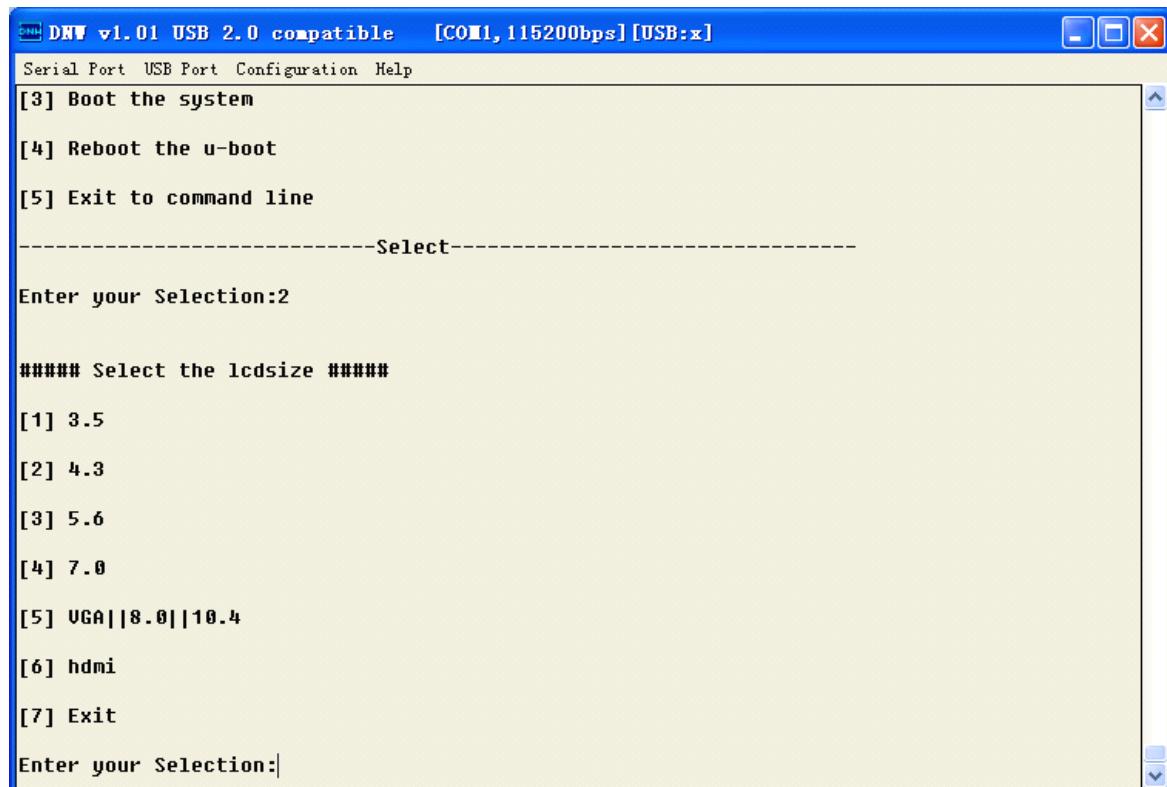
步骤1. 保证系统可以启动飞凌的uboot。如果不能启动，请参看前面的烧写步骤。

步骤2. 先用串口线连接好开发板COM0与PC机的串口，打开并设置DNW软件（附录：dnw软件的使用简便教程）。

步骤3. 然后给开发板上电，等到出现延时1秒启动系统时，在DNW软件中按PC键盘的空格键使开发板停留在uboot状态。因为停留时间只有1秒，所以需要很快的按下空格键。如图：



输入2，会进入屏幕尺寸设置界面，如下图所示：



根据您的 LCD 大小选择不同的选项即可。

4-4 SD/MMC卡驱动测试

开发板支持 SD 卡热插拔。插上 SD 卡后系统会自动将其挂载到 /sdcard 目录。

目前测试支持 8G SD 卡，8G 以上并未测试。

物理连接如图：



同时终端会打印关于 SD 卡的信息，由于存在很多种卡，显示的信息可能会有差别：

```
s3c6400_setup_sdhci_cfg_card: CTRL 2=c0004100, 3=80808080
s3c6400_setup_sdhci_cfg_card: CTRL 2=c0004100, 3=00008080
mmc0: new high speed SDHC card at address d3f6
mmcblk0: mmc0:d3f6 SD08G 7.42 GiB
mmcblk0: p1
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will
be case sensitive!
```

查看 SD 卡中的文件，命令如下，效果如下图：

`#ls /sdcard`

```
[root@FORLINX6410]# ls /sdcard
5555           com31           nand
88KEY.EXE      loopback       u-boot.bin
MP3            mcp251x.ko     zImage
```

以上是命令行测试 SD 卡的方法，当然如果您不喜欢使用命令行测试，也可以使用图形界面测试程序来看看 SD 卡里面的内容，如下图所示：



ForlinxTest 程序组里面有一个文件浏览器，您可以使用文件浏览器软件来浏览您的 NandFlash, SD 卡, U 盘里面的文件，当我们把 sd 卡插入开发板中，使用文件浏览器打开 `sdcard` 目录，里面存放了 SD 卡的文件，当我们使用 U 盘时，使用文件浏览器打开 `udisk` 目录，这个目录里面显示的是 U 盘中的文件。如图所示，我们使用的是 SD 卡：



双击 **sdcard** 目录,可以看到 SD 里面的内容。



4-5 USB鼠标输入和触摸输入的切换

本系统暂时不能同时支持 USB 鼠标和触摸同时输入。

切换 USB 鼠标输入的方法： 在终端中输入命令 **#mouseinput**，如图：

```
[root@FORLINX6410]# mouseinput
[root@FORLINX6410]# mouse...
[root@FORLINX6410]#
```

切换触摸输入的方法：

在终端中输入命令 **#touchinput**，如图：

```
[root@FORLINX6410]# touchinput  
[root@FORLINX6410]# touch...  
[root@FORLINX6410]#
```

touchinput 和 mouseinput 命令实质上是放在根文件系统/sbin 目录中的脚本，路径如图：

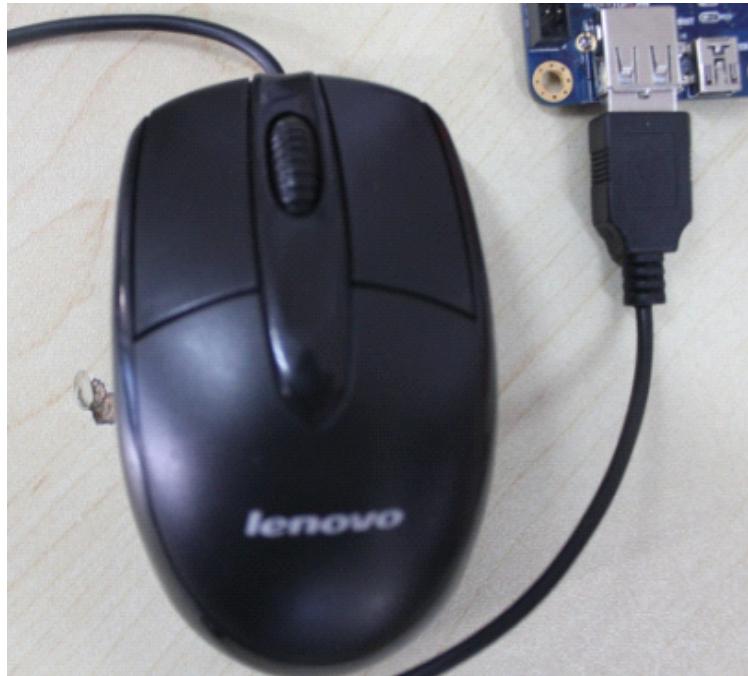
```
[root@FORLINX6410]# pwd  
/sbin  
[root@FORLINX6410]# ls  
adjtimex      halt          makedevs      setconsole  
arp           hparm         man           setfont  
blkid         httpd        mdev          setlogcons  
brctl         hwclock     mkfs.minix    shutdown  
chpasswd      ifconfig     mkswap       slattach  
chroot        ifdown       modprobe    start-stop-daemon  
crond         ifenslave   mouseinput  sulogin  
depmod        ifup          nameif      svlogd  
devmem        inetd        pivot_root  swapoff  
dhcprelay     init          popmaildir swapon  
dnssd         insmod      poweroff    switch_root  
fakeidentd   iwconfig    raidautorun sysctl  
fbset          iwlist      rdate       syslogd  
fbsplash     klogd        rdev        telnetd  
fdisk         loadfont    readprofile touchinput  
findfs        loadkmap    reboot      udhcpc  
freeramdisk   logread     rmmod       udhcpd  
fsck          losetup     route      vconfig  
fsck.minix    lpd          runlevel    watchdog  
getty         lsmod       sendmail    zcip
```

4-6 USB HOST 接口测试

4-6-1 USB鼠标

- 步骤 1. 根据 4-4 的方法切换成 USB 鼠标输入。
步骤 2. 插入 USB 鼠标到开发板底板的 USB Host 口。

物理连接如图：



连接后串口终端会打印 USB 鼠标的信息，因为鼠标芯片种类很多，所以信息不一定完全一样。信息如图：

```
[root@FORLINUX6410]# usb 1-1: new low speed USB device using s3c2410-ohci and
address 4
usb 1-1: New USB device found, idVendor=1c4f, idProduct=0003
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: U+P Mouse
usb 1-1: Manufacturer: SIGMACH1P
input: SIGMACH1P U+P Mouse as /class/input/input5
generic-usb 0003:1C4F:0003.0004: input: USB HID v1.10 Mouse [SIGMACH1P U+P
Mouse] on usb-s3c24xx-1/input0
```

4-6-2 USB键盘

飞凌在内核中已经添加了对 USB 键盘的支持，插入 USB 键盘，串口会出现下图所示提示信息，这时 USB 键盘就可以使用了。

物理连接如图：



连接后串口终端会打印 USB 鼠标的信息，因为鼠标芯片种类很多，所以信息不一定完全一样。信息如图：

```
[root@FORLINUX6410]# usb 1-1: new low speed USB device using s3c2410-ohci and address 6
usb 1-1: New USB device found, idVendor=1c4f, idProduct=0002
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: USB Keykoard
usb 1-1: Manufacturer: USB
input: USB USB Keykoard as /class/input/input8
generic-usb 0003:1C4F:0002.0007: input: USB HID v1.10 Keyboard [USB USB
Keykoard] on usb-s3c24xx-1/input0
```

4-6-3 挂载U盘

开发板支持 U 盘热插拔。插上 U 盘后系统会自动将其挂载到 /udisk 目录。

目前 U 盘测试支持到 32G，32G 以上并未测试。

物理连接如图：



同时终端会打印关于 U 盘的信息，由于存在很多种 U 盘，显示的信息可能会有差别：

```
[root@FORLINX6410]# usb 1-1: new full speed USB device using s3c2410-ohci and address 8
usb 1-1: New USB device found, idVendor=058f, idProduct=6387
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-1: Product: F3
usb 1-1: Manufacturer: EAGET
usb 1-1: SerialNumber: 3BFD4ED7
scsi1 : usb-storage 1-1:1.0
scsi 1:0:0:0: Direct-Access      EAGET      F3          8.07 PQ: 0 ANSI: 2
sd 1:0:0:0: Attached scsi generic sg0 type 0
sd 1:0:0:0: [sda] 62283776 512-byte logical blocks: (31.8 GB/29.6 GiB)
sd 1:0:0:0: [sda] Write Protect is off
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Assuming drive cache: write through
  sda: sda1
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Attached SCSI removable disk
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will
be case sensitive!
```

查看 U 盘中的文件，命令如下，效果如下图：

```
#ls /udisk
```

```
[root@FORLINX6410]# ls /udisk
(USB- SERIEL2303)
111.bmp
111.tmp
123_backup.camproj
128-256
2011-01-20 FL6410_256_2G
```

以上是使用命令行测试 U 盘挂载的方法，您也可以使用图形界面的方式来测试 U 盘。把 U 盘插入开发板的 USB 接口，使用 Qtopia2.2.0 桌面环境里面的文件浏览器，查看 U 盘里面的文件，具体请参考 SD 卡测试一节。

4-7 以太网驱动测试及相关服务

4-7-1 网络相关配置

本节测试中，网络环境如下：

网络连接方式：开发板直接连接能够上网的路由器。

开发板示例 IP：192.168.0.232

路由器 IP：192.168.0.201

子网掩码：255.255.255.0

每个开发板的网络使用环境未必相同，本小节使用上述网络环境作为演示。实际使用中，请按照实际网络环境自行进行配置。

- 命令行设置 IP 地址

```
#ifconfig eth0 192.168.0.232 (将 eth0[dm9000]设置 IP 为 192.168.0.232)
#ifconfig eth0 up (将 eth0[dm9000]使能)
#ifconfig (查看当前网络状况)
```

```
[root@FORLINX6410]# ifconfig eth0 192.168.0.232
[root@FORLINX6410]# ifconfig eth0 up
[root@FORLINX6410]# ifconfig
eth0      Link encap:Ethernet HWaddr 08:90:90:90:90:90
          inet addr:192.168.0.232 Bcast:192.168.0.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:3402 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:371963 (363.2 KiB) TX bytes:42 (42.0 B)
          Interrupt:108 Base address:0x6000

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

如果您的开发板与路由器连接，且路由器支持 DHCP 自动 IP 地址分配，可以在 DNW 或者超级终端里面输入：

udhcpc -i eth0

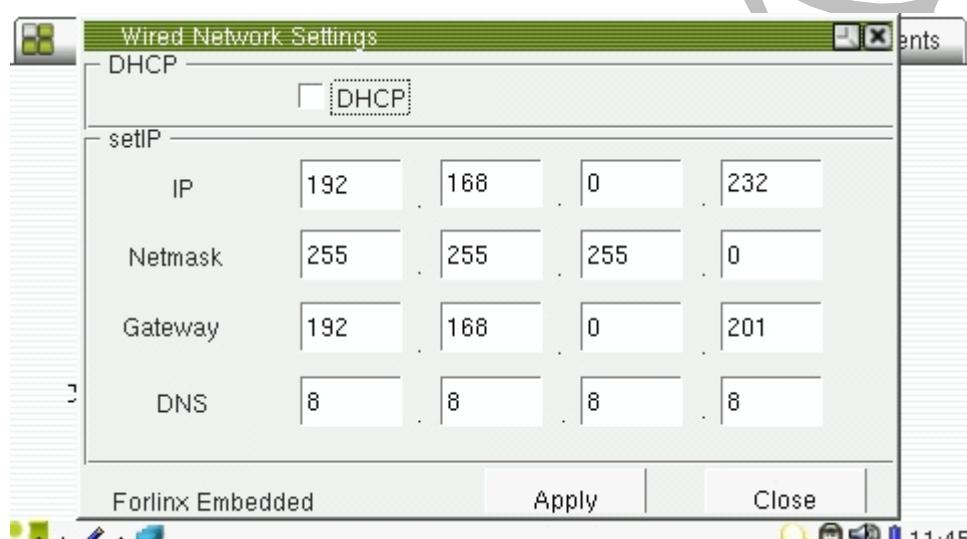
命令，用来动态获取 IP 地址，-i 参数用来指定网卡名称，飞凌开发板有线网络的网卡名称为 eth0.

- 图形界面设置 IP 地址

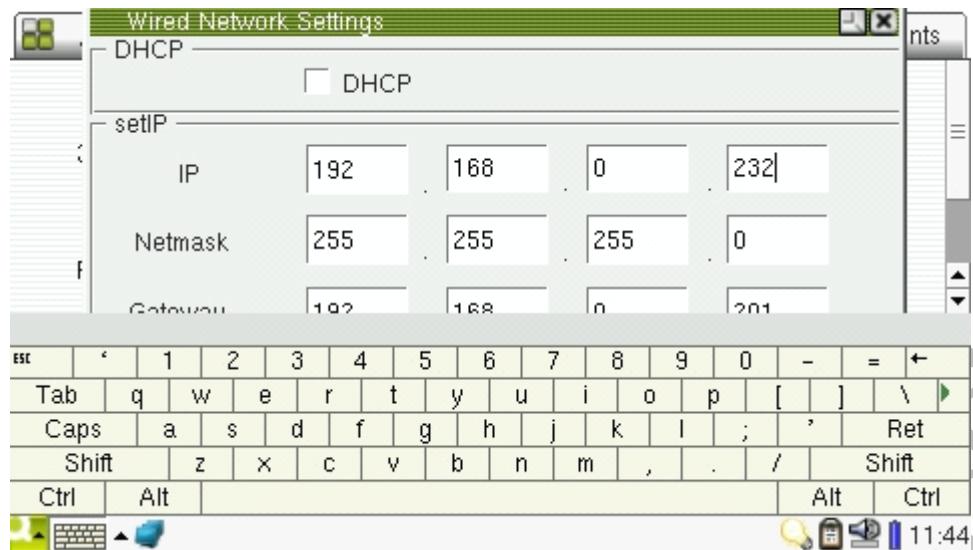
Qtopia2.2.0 桌面环境，ForlinxTest 程序组中有一个 IP 地址设置软件，该软件可以设置固定 IP 地址，也可以从路由器动态获取 IP 地址，是您网络应用的好帮手，如图所示：



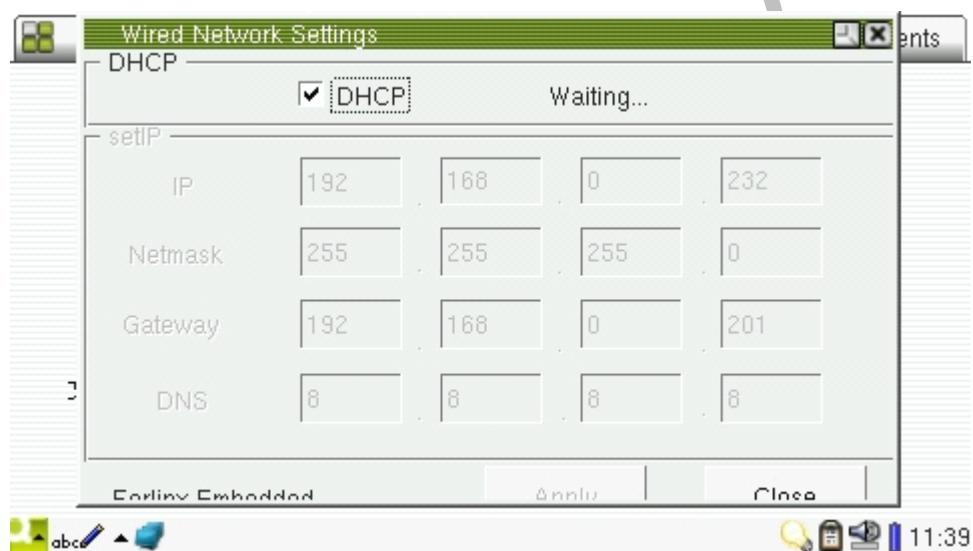
双击 setIP 图标:



您可以使用软键盘输入 IP 地址，当然也可以使用 USB 键盘，下图使用的是软键盘：

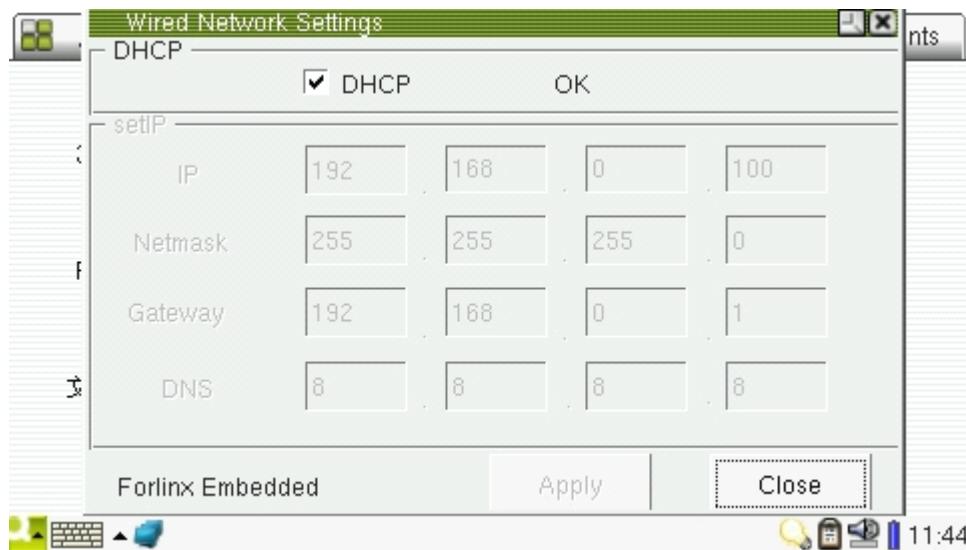


您可以使用动态 IP 获取功能，当然需要路由器支持 DHCP。单击 DHCP 按钮，路由器开始分配 IP 地址，



成功分配 IP 后，会提示 OK，且会显示分配到的 IP 地址，子网掩码，网关，DNS 等具体

的网络参数：



4-7-2 使用ping命令测试网络

- 可以使用 ping 命令测试网络是否连通：

```
#ping 192.168.0.201 -s 10000
```

其中 10000 为数据包大小；按键盘的 Ctrl + C 或者 Ctrl + Z 退出。

```
[root@FORLINX6410]# ping 192.168.0.201
PING 192.168.0.201 (192.168.0.201): 56 data bytes
64 bytes from 192.168.0.201: seq=0 ttl=64 time=1.788 ms
64 bytes from 192.168.0.201: seq=1 ttl=64 time=0.769 ms
64 bytes from 192.168.0.201: seq=2 ttl=64 time=0.761 ms
```

4-7-3 浏览网页

设置好网络环境后，点击桌面“ForlinxTest”面板上的“Web Browser”图标可浏览网

页，位置如图：



打开 google 主页，如图：



浏览器源码在光盘中的路径：

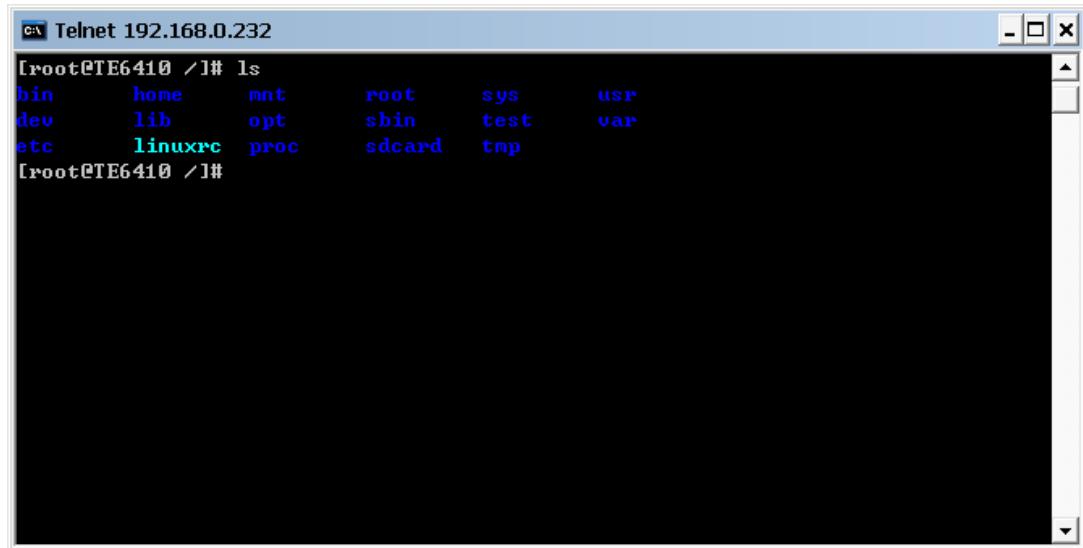
基础光盘\Linux-3.0.1\apptest\QT ApplicationTest\konqueror

4-7-4 Telnet服务

OK6410 开发板在 /etc/init.d/rcS 脚本文件中已经启动了 telnet 服务，设置好 IP 地址后就可以作为一台 telnet 服务器了。

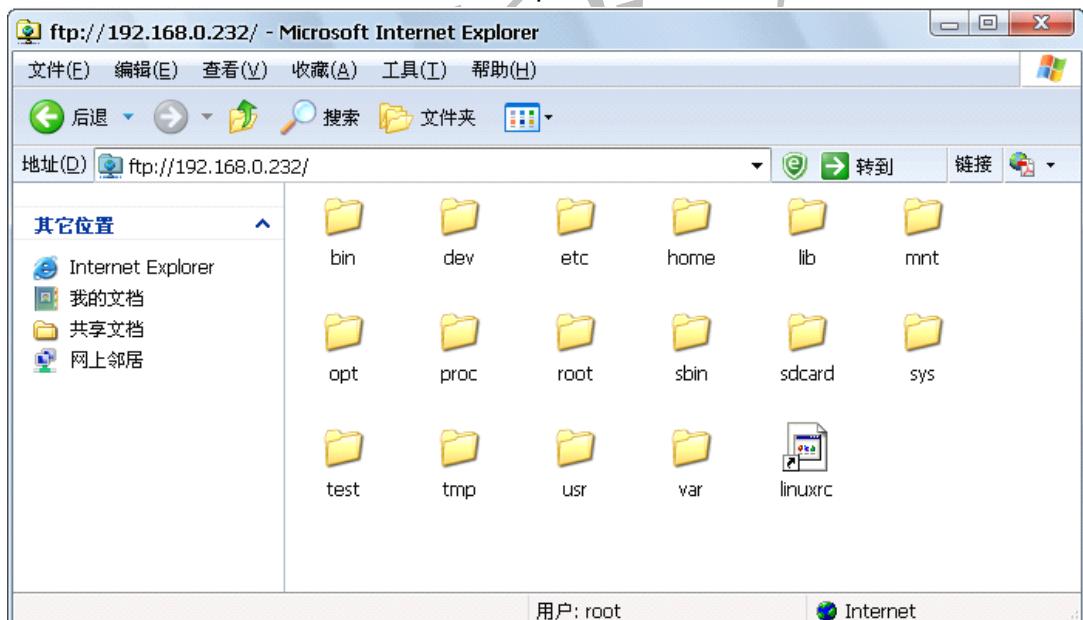
例如，开发板 IP 地址为 192.168.0.232，在 Windows 的命令窗口下输入 telnet

192.168.0.232 ,用户名输入 root , 密码为空 (开始->运行, 输入 cmd 回车, 即可进入 Windows 的命令窗口)



4-7-5 FTP服务

系统启动时已经自动启动了ftp 服务, 可在电脑上用ftp 软件访问, 用户名root , 密码为空; 下图是在 PC 机上用 IE 访问 ftp 的截图:



使用 FTP 功能可以实现开发板与 PC 之间传输文件。

4-7-6 Web服务

我们在开发板上移植了一个webserver: boa ; boa webserver 是一个小巧高效的web服务器, 可运行在Unix 或Linux 平台, 支持CGI , 源代码开放; 是一个非常适合于嵌入式系统的单任务http 服务器。

系统启动时已经自动启动了ftp 服务，在IE 中输入开发板的IP 地址即可浏览开发板 webserver 中的网页。下图是在IE 中浏览的截图：



4-7-7 NFS挂载网络文件系统

■ 准备 NFS 文件系统目录

1. 准备NFS文件系统目录

启动nfs 服务之前，必须在Ubuntu 上准备好NFS 共享目录。

例如，我们采用Ubuntu 的“/forlinx/root”作为NFS 共享目录，就需要将用户基础资料光盘中的 FileSystem-Yaffs2.tar.gz 压缩文件拷贝到这个目录下，然后解压缩，得到根文件系统所需要的目录。

在Ubuntu 上打开一个终端，输入以下命令：

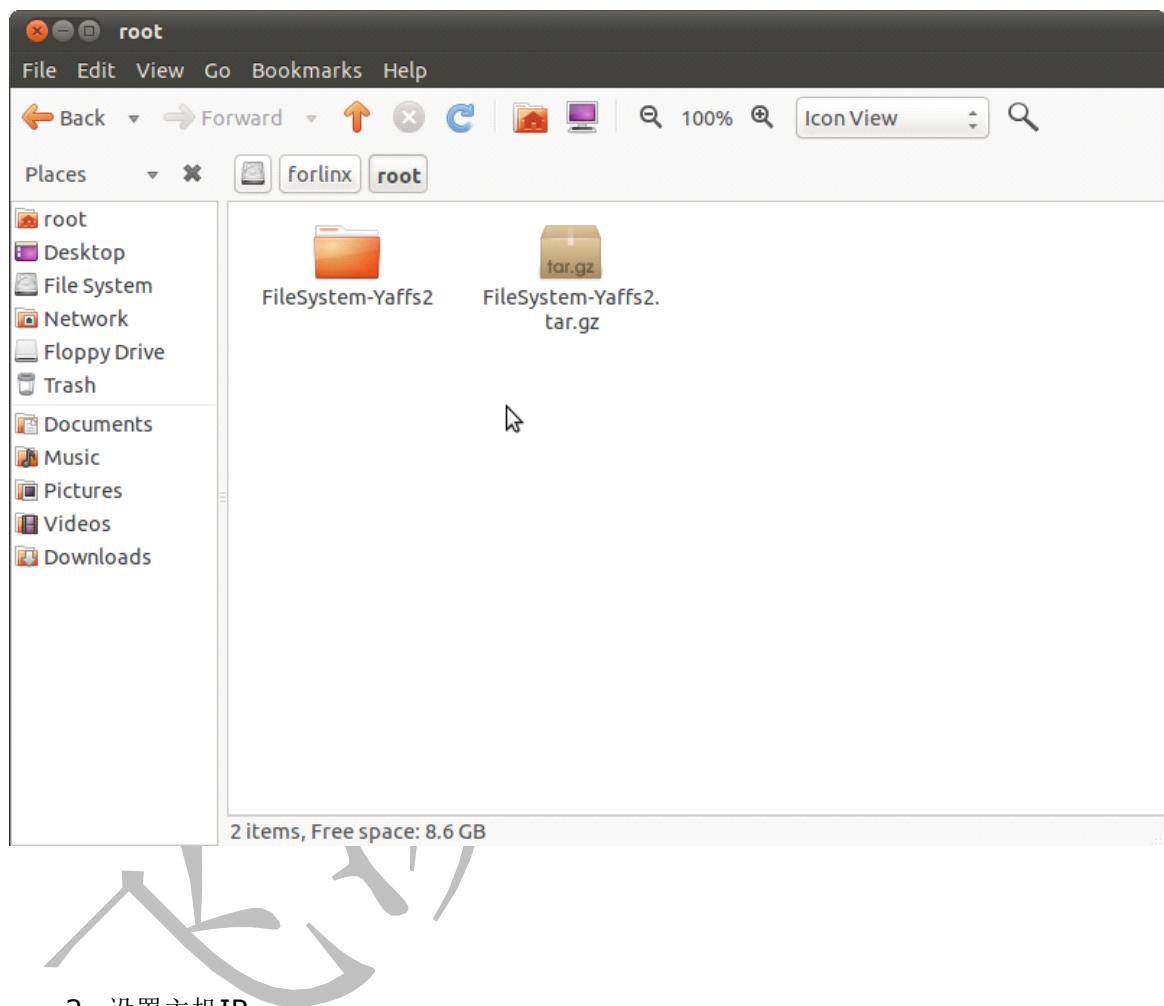
```
#mkdir /forlinx/root
```

将FileSystem-Yaffs2.tar.gz 文件拷贝到该目录下，解压：

```
#tar -zxf FileSystem-Yaffs2.tar.gz
```

解压完成后如图所示：

如图所示：



2. 设置主机IP

这里我们将Ubuntu 的IP 设置为 192.168.0.231

3. 配置NFS服务

在Ubuntu 上新建一个终端，依次输入以下命令：

```
#sudo apt-get install portmap
```

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

```
#sudo gedit /etc/exports
```

在弹出的文本编辑器中编辑exports 文件，在最后一行添加：

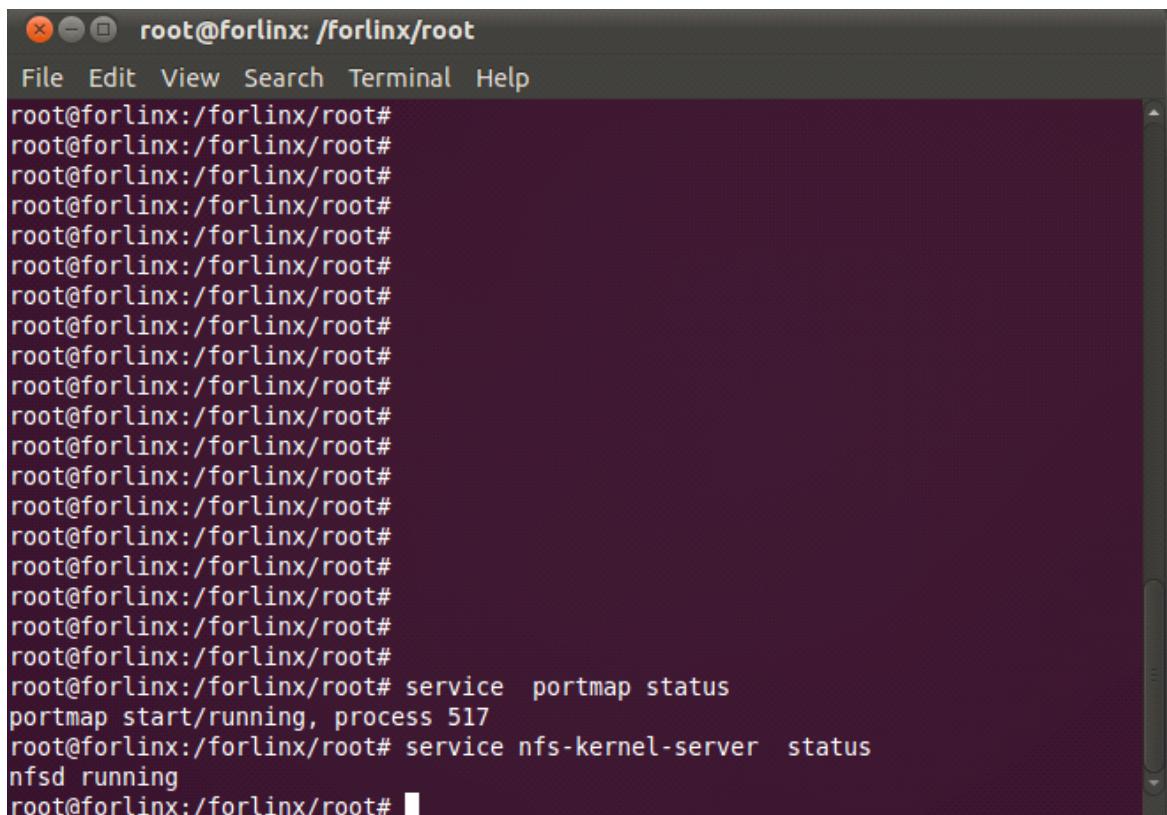
```
/forlinx *(rw,sync,no_root_squash)
```

4. 启动NFS服务

```
#sudo /etc/init.d/portmap restart  
#sudo /etc/init.d/nfs-kernel-server restart
```

5 检查服务是否已经运行

```
#service portmap status  
#service nfs-kernel-server status
```



The screenshot shows a terminal window titled "root@forlinx: /forlinx/root". The window contains the following text:

```
File Edit View Search Terminal Help  
root@forlinx:/forlinx/root#  
root@forlinx:/forlinx/root# service portmap status  
portmap start/running, process 517  
root@forlinx:/forlinx/root# service nfs-kernel-server status  
nfsd running  
root@forlinx:/forlinx/root#
```

从这里我们看到 portmap 和 nfs-kernel-server 服务已经运行了。

■ 挂载根文件系统到宿主机

 在u-Boot 命令行下输入以下命令设置U-boot 启动参数:

```
#setenv  
bootargs"root=/dev/nfsnfsroot=192.168.0.231:/forlinx/root/FileSystem-Yaffs2  
ip=192.168.0.232:192.168.0.231:192.168.0.201:255.255.255.0:witech.com.  
cn:eth0:off console=ttySAC0,115200"
```

保存:

```
#saveenv
```

注意：`setenv`和`bootargs` 之间不是回车，而是空格。

另外，我们提供的光盘资料中有`nfs-tftp.txt` 文本文件，里面有NFS根文件系统的设置方法，您不需要自己输入了，**复制**`nfs.txt`文件中的 **NFS FileSystem** 下面的设置，**粘贴到Uboot中即可，里面也有NandFlash 本地 FileSystem**，您可以在网络文件系统和本地文件系统之间自由的切换，当然您不要忘了 执行 `saveenv` 保存参数设置命令哦。

如下图所示：



```
nfs-tftp.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

IP地址设置：
#设置开发板IP地址:
setenv ipaddr 192.168.0.232

#设置虚拟机IP地址:
setenv serverip 192.168.0.231

#保存设置参数
saveenv

TFTP:
1 Net
setenv bootcmd tftp 20008000 zImage\; bootm 20008000

2 Local
setenv bootcmd nand read C0008000 100000 500000\; bootm C0008000

NFS:
1 NFS FileSystem
setenv bootargs "root=/dev/nfs nfsroot=192.168.0.231:/work-2012/linux \
ip=192.168.0.232:192.168.0.231:192.168.0.201:255.255.255.0:witech.com.cn:eth0:off \
console=ttySAC2,115200"

2 NandFlash FileSystem
setenv bootargs "root=/dev/mtdblock2 rootfstype=yaffs2 console=ttySAC2,115200"
```

重新启动开发板，Linux 内核启动后会自动挂载NFS 文件系统。

在这，解释一下`bootargs` 参数中IP，以上述设置为例。在实际使用过程当中，请以实际网络环境进行修改：

192.168.0.231 PC 端Ubuntu 的IP

192.168.0.232 开发板IP，开发板的IP必须和 PC端的IP在同一个网段，这里都是0网段

192.168.0.201 网关

255.255.255.0 子网掩码

nfs 挂载成功，需要开发板网络设置、PC Linux 网络设置、硬件网线连接、开发板mount 这几部分都没有问题。如果没有成功挂载，需要从这几部分查找原因。如果是使用虚拟机安装 Linux，在挂载 nfs 的时候，建议关闭 Windows 的杀毒软件和防火墙。

■ 挂载目录文件到宿主机

步骤 1. 根据实际情况，设置 PC 端 Linux 的 nfs 服务器。

这里我们假设/etc(exports 的内容为 “/ *”

/ 代表 PC Linux 根目录和子目录都可以被挂载。

* 代表挂载的时候权限为最大

假设 PC Linux 的 IP 设置为 192.168.0.1

步骤 2. 正确设置开发板的 IP 等网络环境，注意要使开发板的 IP 和 PC Linux 在同一网段。

步骤 3. 关闭 PC 所有的杀毒软件和防火墙（尤其是 xp 虚拟 linux 时，xp 下的防火墙和杀毒软件）。如不关闭，有可能无法挂载。

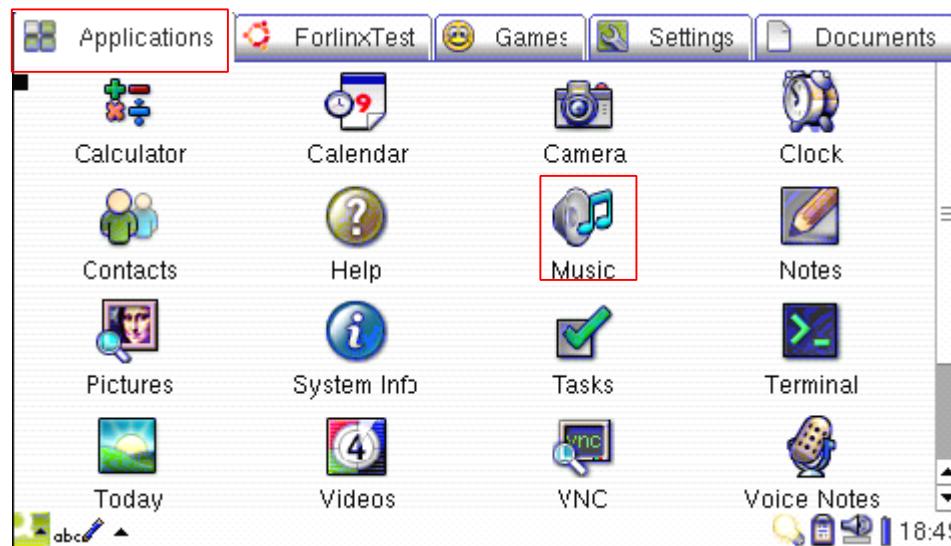
步骤 4. 挂载 nfs 到开发板的/temp。

```
#mount -t nfs -o nolock 192.168.0.1:/mnt /temp
```

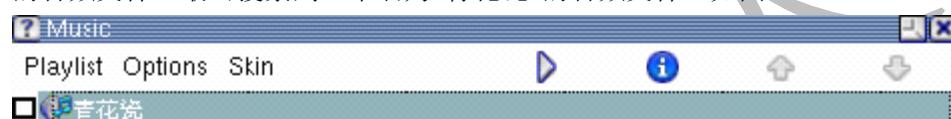
命令执行成功后，开发板的 /temp 目录就和 Linux 宿主机的 /mnt 目录建立了共享关系，在超级终端或者 DNW 中执行 ls -al /temp，可以看到宿主机/mnt 文件目录下面的所有内容。另外您可以在 PC Linux 上编译应用程序，放到 PC 机的/mnt 目录，在超级终端或者 DNW 中执行 cd / temp 命令进入到 temp 目录下面，执行应用程序，这样会加快您的研发调试进度。

4-8 音频播放测试

在 qtopia 桌面上打开 application 选项标签，可以看到“Music”图标，“Music”是音频播放器的应用程序，如图：



点击“Music”播放器，播放器自动搜索开发板中/root/QtopiaHome/Documents/目录下的音频文件。最终搜索到一个名为“青花瓷”的音频文件，如图：



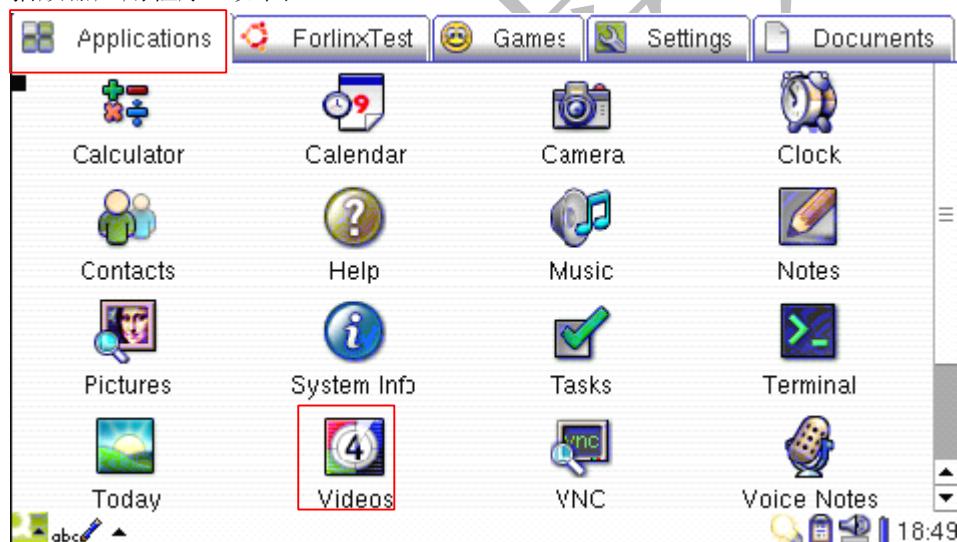
点击文件名即可播放该音频文件。所以如图：



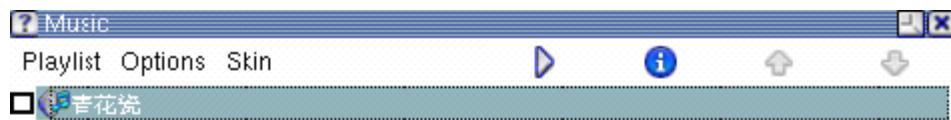
如果需要退出，可以点击屏幕右上角的X，红色框中的按钮可以调节音量的大小。

4-9 视频播放测试

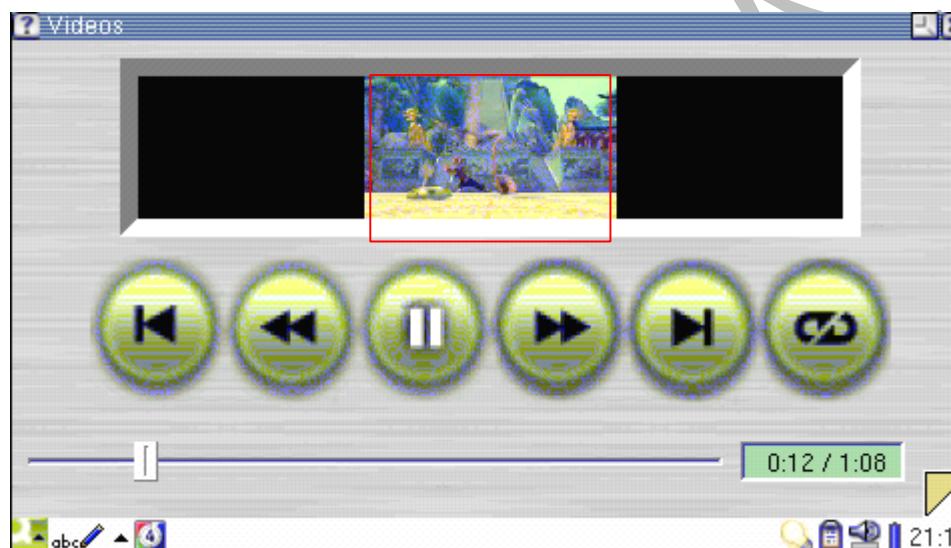
在 qtopia 桌面中打开 application 选项标签，可以看到“Videos”图标，“Videos”是视频播放器应用程序，如图：



点击“Videos”播放器，播放器自动搜索开发板中/root/QtopiaHome/Documents/目录下的视频文件。最终搜索到一个名为“panda”的音频文件，如图：



点击文件名即可播放该视频文件。点击中间视频部分可以最大化视频。所以如图：



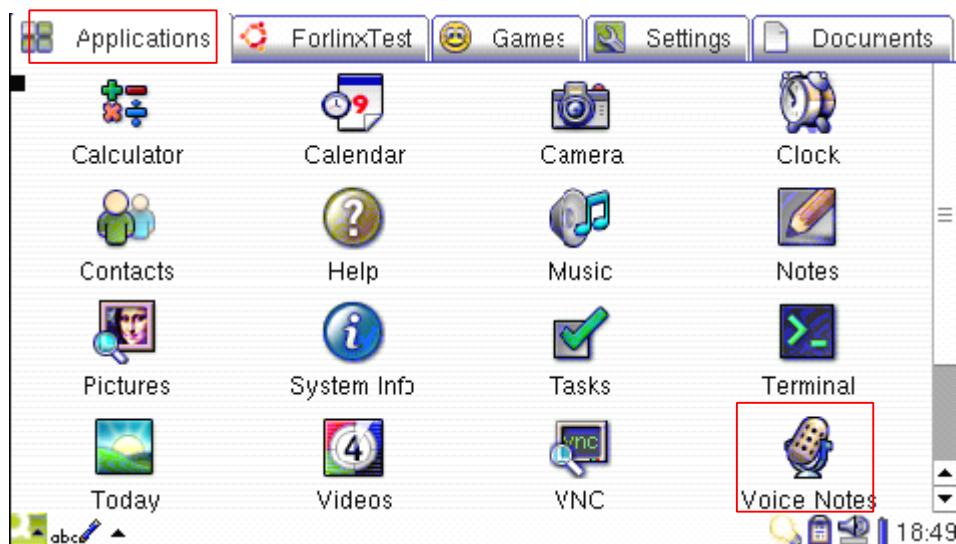
如果需要退出，可以点击屏幕右上角的 \times 。

4-10 录音测试

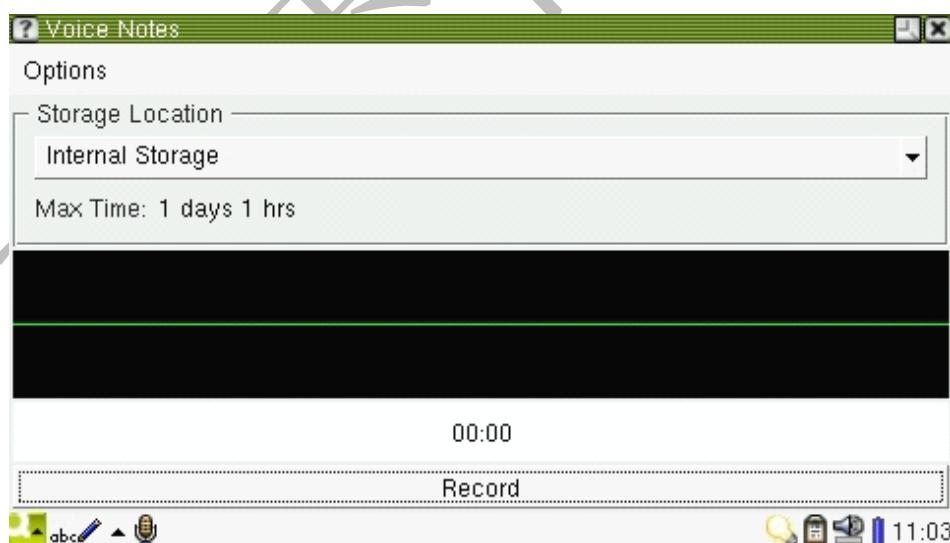
录音测试有两种方法：语音记事本测试和 ben 测试程序。

4-10-1 语音记事本

在 qtopia 桌面中打开 application 选项标签，可以看到“Voice Notes”图标，“Voice Notes”是记录语音用的程序，如图：



打开之后点击 new，创建新语音文件，如图：



点击 Record 即可进行录音，点击 stop 停止录音，停止录音之后会看到 new 下面产生了新的文件，点击该文件，即可播放录音。

4-10-2 ben 测试程序

首先将耳机和麦克风插到开发板上相应位置。

ben 可执行文件在/usr/bin/下面，测试步骤如下：

```
[root@FORLINX6410]# cd /usr/bin/  
[root@FORLINX6410]# ./ben  
Saying:  
  
saying mun 48000Said:  
  
saying mun 48000Saying:
```

当出现“Saying：”时是录音状态，等录音完毕，按回车键，出现“saying mun 48000Said：”，此时耳机里传出录音。

ben 程序源码路径：

基础光盘\ Linux-3.0.1\apptest\ben_test\ben.c

4-11 采样测试

步骤 1. 在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“ADTest”图标，“ADTest”是采样测试应用程序，如图：



步骤 2. 点击“ADTest”，开始显示 AD 转换值：



步骤 3. 触笔点击屏幕，会显示-1 值，当触笔离开屏幕，又恢复原来的值。如果出现-1 值，说明触摸和采样可以同时使用。



AD_Test 应用程序源码路径：

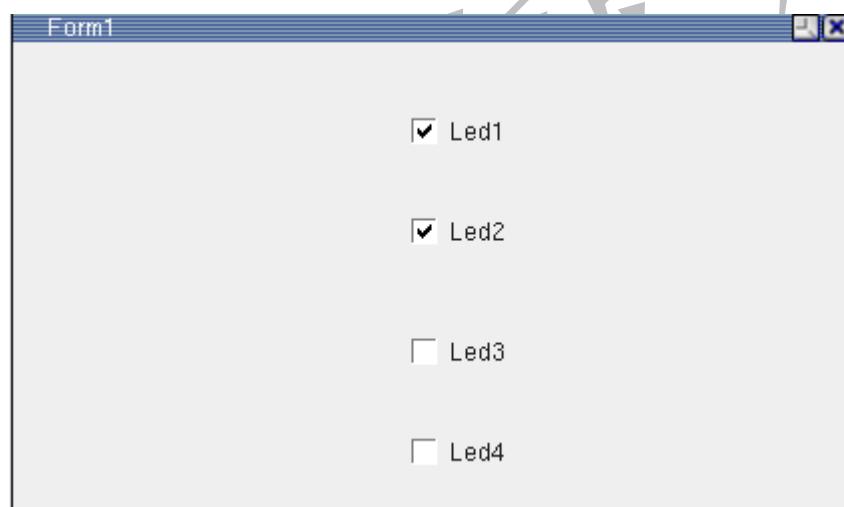
基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\ADTest

4-12 LED测试

在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“LedTest”图标，“LedTest”是 LED 测试应用程序，如图：



点击“LedTest”，显示 LED 控制面板。当勾选 LED 后，对应的 LED 会点亮；取消对 LED 的勾选，对应的 LED 会熄灭。



每次操作，串口终端会显示当前 LED 的显示状态，上图中 LED 状态对应下面的串口信息：

```
leds: 0 1  
leds: 1 1  
leds: 2 0  
leds: 3 0
```

LED_Test 应用程序源码路径：

基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\ledTest

4-13 蜂鸣器测试

在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“PWMTest”图标，“PWMTest”是开发板蜂鸣器的测试程序，如图：



点击“PWMTest”，显示 PWM 控制面板。点击“StartBuzzer”蜂鸣器开始鸣叫；点击“StopBuzzer”蜂鸣器停止鸣叫；在打开蜂鸣器之后，利用下图红色部分可以调节频率，包括粗调和微调。



PWMTest 应用程序源码路径：

基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\PWMTest

4-14 TV输出

OK6410 开发板带一路 TV-Out, 这一路是 CPU 自带的 TV-Out。TE6410 有两路 TV-Out, 一路是 CH7026 转 TV-Out, 另一路是 CPU 自带 TV-Out。这一小节 中只测试 CPU 自带的这一路。

连接方法: 使用 TV-Out 输出线连接开发板的 TV 接口和电视的 TV video in 接口。

启动开发板, 进入 Linux3.0 系统, 在命令行中运行命令 **#tv-out**, 这时就会从电视中看到图形界面。

重启开发板即可恢复到 LCD 输出。

Tv_out 源码目录:

基础光盘\Linux-3.0.1\apptest\tv_out



4-15 红外测试

在文件系统/usr/bin/下面有红外测试程序 irda, 测试步骤如下:

```
[root@FORLINX6410]# pwd  
/opt/Qtopia/bin  
[root@FORLINX6410]# ./ilda
```

运行之后超级终端会不停的打印“1”, 当有红外感应时, 会打印“0”。
该程序只是测试硬件电平, 没有软件协议。

ilda 源码目录:

基础光盘\Linux-3.0.1\apptest\ilda_test\ilda_test.c

4-16 温度传感器测试

在文件系统/usr/bin/下面有温度传感器测试程序 temp_test, 测试步骤如下:

```
[root@FORLINX6410]# cd /usr/bin/  
[root@FORLINX6410]# ./temp_test  
Open Device DS18B20 successed.  
34.10C  
34.10C  
34.11C  
35.3C  
35.4C  
35.3C
```

注意: 目前打印的温度值会比正确值高出一些, 我们还在进一步调试中。

温度传感器源码目录:

基础光盘\Linux-3.0.1\apptest\temp_est\temp_test.c

4-17 串口测试

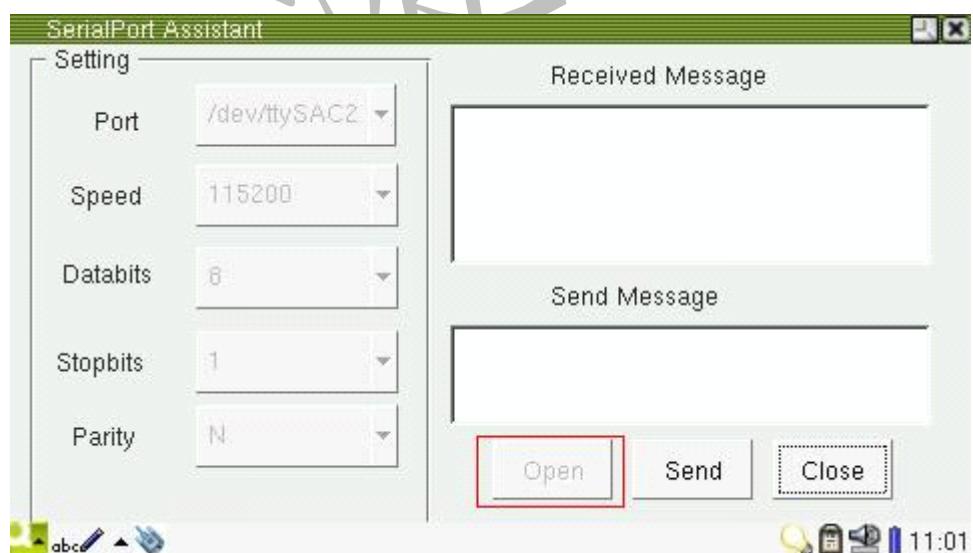
通过串口测试程序可以设置其波特率、数据位、停止位以及奇偶校验。

步骤 1.开发板连接。开发板 COM2 通过直连串口线和电脑串口相连，在 PC 机上打开超级终端，设置好属性。

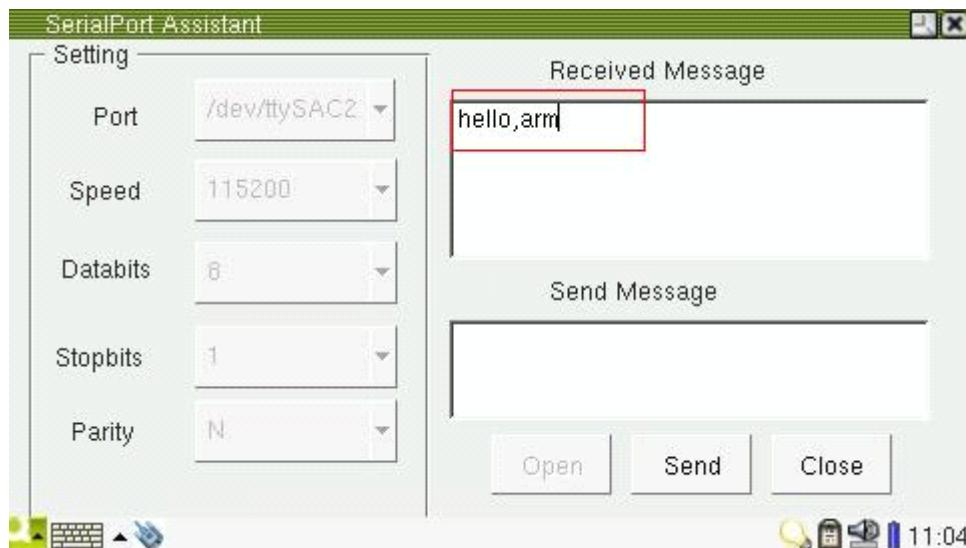
步骤 2.打开软件。在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“serialport”图标，“serialport”是开发板串口的测试程序，如图：



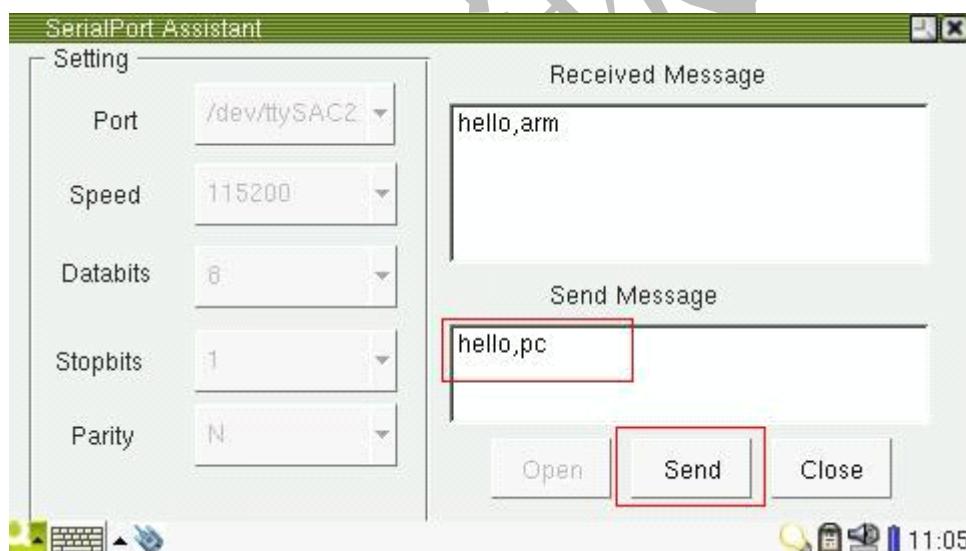
步骤 3.打开串口。待设置好串口属性之后，单击 Open 按钮，即可打开串口。此处我连接的是 COM2，所以选择 /dev/ttYSAC2。设置好串口属性，属性要和 PC 机上超级终端一样。



步骤4.PC机向开发板发送数据。在PC超级终端里边写入"hello,arm",开发板上Received Message下面的框里便会收到PC发送来的信息。



步骤5.开发板向PC机发送数据。在开发板Send Message栏里写上"hello,pc",单击Send按钮发送,在PC机的超级终端里边就会显示出来。



步骤6.退出程序。先点击Close按钮,关闭串口,再点击右上角关闭按钮。

serialport 应用程序源码路径:

基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\serialport

第五章 测试外围模块

5-1 USB 3G 上网卡测试

(1) 测试电信 CDMA2000 模块

步骤 1. 现将 3G 手机卡插好在 USB 3G 模块中。

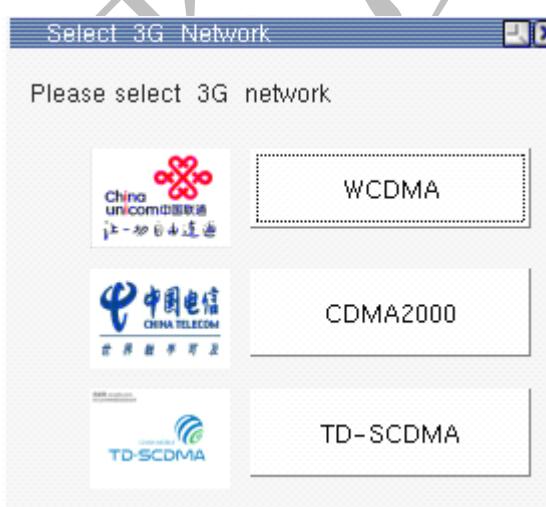
步骤 2. 开发板上电，启动 Linux 系统。

步骤 3. 本小节中是一个 USB 3G 模块测试的示例。由于芯片的不同，所以在您做本实验时，有可能和本小节测试打印数据稍有差别。

在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“3GDialup”图标，“3GDialup”是 USB 3G 模块的应用程序，如图：



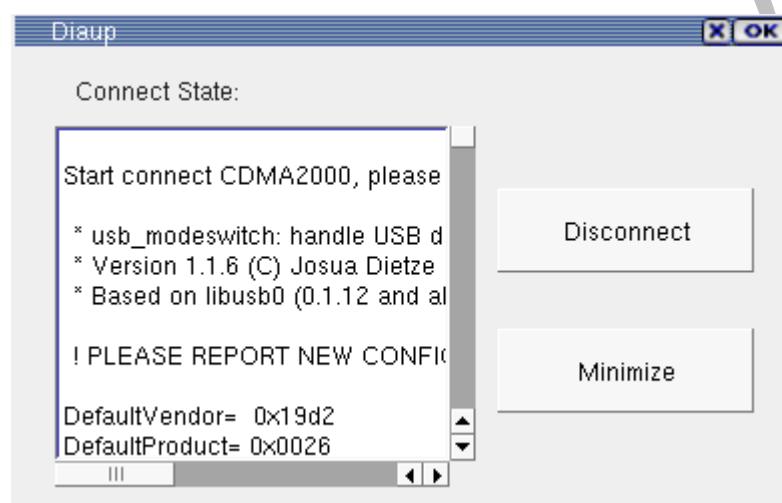
点击“3GDialup”启动应用程序，看到 3G 模块的控制面板：



点击图标中“CDMA2000”按钮。出现 CDMA2000 模块控制面板：



点击“Connect Net”， Connect State 出现连接 CDMA2000 网络的信息，如图：



点击“Minimize”可以最小化 CDMA2000 的控制面板。

此时，已经连接好网络，打开互联网浏览器，打开 google 首页，如图：



目前，飞凌提供的 3G 测试程序支持 Modem 型号：

WCDMA: 飞凌 AD3812;

CDMA2000: ZTE-AC581、ZTE-AC582;

TD-SCDMA: HUAWEI-ET127、ZTE- A356;

以上型号经过测试，其他型号暂未测试。

3GTest 应用程序源码路径：

基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\3GTest

5-2 CMOS摄像头 OV9650 测试

OV9650 摄像头是选配模块。

步骤 1. 断电、连接号飞凌的 OV9650 摄像头到飞凌开发板的 CAM 接口。

请注意：OV9650 模块不能带电插拔。

步骤 2. 开发板上电，启动 Linux 系统。

步骤 3. 运行测试程序的命令：`#testcamera`

摄像头采集的信息会在 LCD 屏显示出来（目前的测试程序是 4.3 寸测试程序，您可以根据自己的实际情况修改源码，以适应自己的屏幕）。



Testcamera 源码在光盘的目录：基础光盘\Linux-3.0.1\apptest\摄像头测试。

CMOS 摄像头的详细测试可以参考多媒体测试相关章节。

5-3 SDIO WIFI

SDIO WIFI 无线局域网卡是选配模块。

SDIO WIFI 无线局域网卡是选配模块。连接方法如图：



步骤 1. 断电、连接号飞凌的 SDIO WIFI 到飞凌开发板的 SD 接口。

步骤 2. 开发板上电，启动 Linux 系统。

步骤 3. 本小节中是一个 SDIO WIFI 连接路由的示例。由于网络环境的不同，所以在您做本实验时，请根据实际情况进行设置。

注意： 新版本的 SDIO WiFi 模块使用圆圈作为起始 PIN 标记，替代了小三角。

执行下面的命令连接路由器。

#ifconfig -a (检测开发板所有网卡状况)

```
[root@FORLINX6410]# ifconfig -a
eth0      Link encap:Ethernet HWaddr 08:90:90:90:90:90
          inet addr:192.168.0.232 Bcast:192.168.0.255 Mask:255.255.255.0
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
          Interrupt:108 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet HWaddr 00:27:13:ED:27:A0
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

#ifconfig eth0 down (关闭 dm9000 网卡)

[root@FORLINX6410]# ifconfig eth0 down

#ifconfig wlan0 up (启动 SDIO WIFI)

[root@FORLINX6410]# ifconfig wlan0 up

#iwlist wlan0 scan (使用 SDIO WIFI 扫描无线网络设备)

```
[root@FORLINX6410]# iwlist wlan0 scan
wlan0      Scan completed :
Cell 01 - Address: 00:21:27:65:77:5E
          Frequency:2.437 GHz (Channel 6)
          Quality=65/70  Signal level=-45 dBm
          Encryption key:on
          ESSID:"TP-LINK_65775E"
          Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                     12 Mb/s; 24 Mb/s; 36 Mb/s
          Bit Rates:9 Mb/s; 18 Mb/s; 48 Mb/s; 54 Mb/s
          Mode:Master
          Extra:tsf=000000045d2e3c9b
          Extra: Last beacon: 590ms ago
          IE: Unknown: 000E54502D4C494E4B5F363537373545
          IE: Unknown: 010882848B960C183048
          IE: Unknown: 030106
          IE: Unknown: 2A0100
          IE: Unknown: 32041224606C
          IE: Unknown: DD0900037F01010008FF7F
          IE: Unknown:
DD1A00037F030100000000212765775E02212765775E64002C010808
```

```
#ifconfig wlan0 192.168.0.232 (设置 SDIO WIFI 的 IP)  
[root@FORLINX6410]# ifconfig wlan0 192.168.0.232
```

```
#iwconfig wlan0 essid "TP-LINK_65775E" (设置 essid)  
[root@FORLINX6410]# iwconfig wlan0 essid "TP-LINK_65775E"
```

```
#iwconfig wlan0 key "123456789" (设置路由器访问密码)  
[root@FORLINX6410]# iwconfig wlan0 key "123456789"
```

```
#route add default gw 192.168.0.201 (设置网关)  
#ping 192.168.0.201 (ping 网关)  
  
[root@FORLINX6410]# route add default gw 192.168.0.201  
[root@FORLINX6410]# ping 192.168.0.201  
PING 192.168.0.201 (192.168.0.201): 56 data bytes  
64 bytes from 192.168.0.201: seq=1 ttl=64 time=3.806 ms  
64 bytes from 192.168.0.201: seq=2 ttl=64 time=3.051 ms  
64 bytes from 192.168.0.201: seq=3 ttl=64 time=3.981 ms  
64 bytes from 192.168.0.201: seq=4 ttl=64 time=3.013 ms  
64 bytes from 192.168.0.201: seq=5 ttl=64 time=3.051 ms  
64 bytes from 192.168.0.201: seq=6 ttl=64 time=3.037 ms
```

本小节测试的路由器可以访问互联网，所以可以使用 QTOPIA 浏览器上网冲浪。

5-4 串口扩展板

串口扩展板是选配模块，专门用于 OK6410 开发板，TE6410 不需要此扩展板。

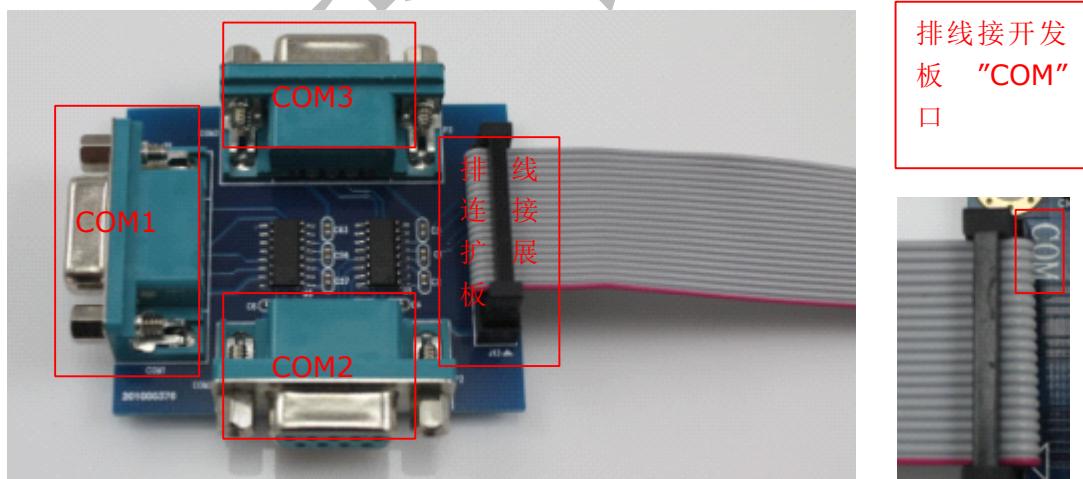
扩展板主要功能：

1. 物理接口转换。将 20pin 的排线接口转换为 DB9 母座接口。可以非常方便的使用串口线连接开发板和外围串口设备。
2. 电气电平接口转换。扩展板使用两片 ttl 转 rs232 电平芯片，将 S3C6410 的 TTL 电平转换为 RS232 电平。其中，扩展板上的 COM1 是五线串口，COM2、COM3 是三线串口。您可以根据您的实际情况，来连接您的外围设备。

串口扩展板三个串口的线序如下：

	COM1	COM2	COM3
1	GND	GND	GND
2	TXD1	TXD2	TXD3
3	RXD1	RXD2	RXD3
4			
5	GND	GND	GND
6	GND	GND	GND
7	CTSN1		
8	RTXN1		
9			

串口扩展板可以直接连接在飞凌 OK6410 开发板的 COM 接口。接法如图。



COM1、COM2、COM3 的设备结点路径分别为：

/dev/ttySAC1

/dev/ttySAC2

/dev/ttySAC3

5-5 4X4 和 8X8 矩阵键盘

飞凌 OK6410 的 A 型板和 B 型板可以外接 4X4 和 8X8 矩阵键盘, TE6410 不支持矩阵键盘。

当前的 zImage 可以识别 4x4 和 8x8 矩阵键盘。您只需正确连接开发板和键盘即可。

连接方法: 使用飞凌提供的排线连接矩阵键盘和开发板 KEY 接口, 如图:



在 QT 中, 打开一个记事本, 按下矩阵键盘的按键, 可在记事本中看到按下按键带来的变化。



注意: 我们的键盘驱动只提供了有限键值, 如需扩展, 请参考驱动程序自行添加。

5-6 USB摄像头

luvcview 是一个开源项目，专注于 UVC 摄像头的测试，只要您的摄像头支持 UVC 驱动，即可使用 **luvcview** 测试程序，如何知道自己的摄像头是不是支持 UVC 驱动呢？在这个网站上查一下，看看自己摄像头的 ID 是不是在支持的列表中，<http://www.ideasonboard.org/uvc/>

我们在 **luvcview** 这个项目的基础上进行了修改，使之能够在 **s3c6410** 平台上运行，修改后的软件采用了 **6410** 特有的硬件空间色彩转换和图像缩放功能，也就是 **post processor** 功能，可以进行图像的全屏显示，且可以抓图保存成图片文件和录制视频文件，详细说明请参考源码目录下面的说明文档。

luvcview 源码位于基础光盘\Linux-3.0.1\apptest\USBCamera- linux+android.rar

5-7 485 接口

TE6410 独有 RS485 接口。内核中已有驱动。针对 **TE6410** 的 RS485 设备节点是 /dev/ttYSAC3

5-8 CAN转接板

TE6410 自带 MCP2515 和 MCP2551 组成的工业级 CAN 收发器。

OK6410 则可以通过飞凌开发的 CAN 扩展板，扩展出和 **TE6410** 同样的 CAN 收发器。

OK6410 连接飞凌开发板和 CAN 扩展板的方法：使用飞凌提供的排线连接 CAN 扩展板和飞凌开发板的用户 IO 接口。

CAN 有很多测试工具。在这里，我们以两块飞凌 **6410** 开发板为例，将 can 口对接，进行测试。

首先介绍一下，飞凌提供的 can 驱动是 **socket can**。应用程序可以像操作一个网络节点一样编程，使用 **socket** 编程即可完成对 can 报文的收发。

步骤 1. 查看 can 节点

```
#ifconfig -a
```

得到 socket can 的信息如下：

```
[root@FORLINX6410]# ifconfig -a
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
           NOARP  MTU:16  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:10
           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

步骤 2. 设置 can 的波特率

```
#up link set can0 up type can bitrate 250000
```

(250000 是波特率, 可以将这个数值在一定范围内取值)

步骤 3. 运行测试程序

其中一块开发板运行 `#can_server` 命令, 然后, 另一块开发板运行 `#can_client` 命令。

这时, 会看到运行 `can_server` 的开发板接收到了 `can_client` 发来的 `can` 报文。

运行 `can_server` 的开发板串口信息显示如下:

```
[root@FORLINX6410]# can_server
Received a CAN frame from interface 0
frame message
--can_id = 123
--can_dlc = 5      接收到的 CAN 信息
--data = hello
Received a CAN frame from interface 2
frame message
--can_id = 123
--can_dlc = 5
--data = hello
```

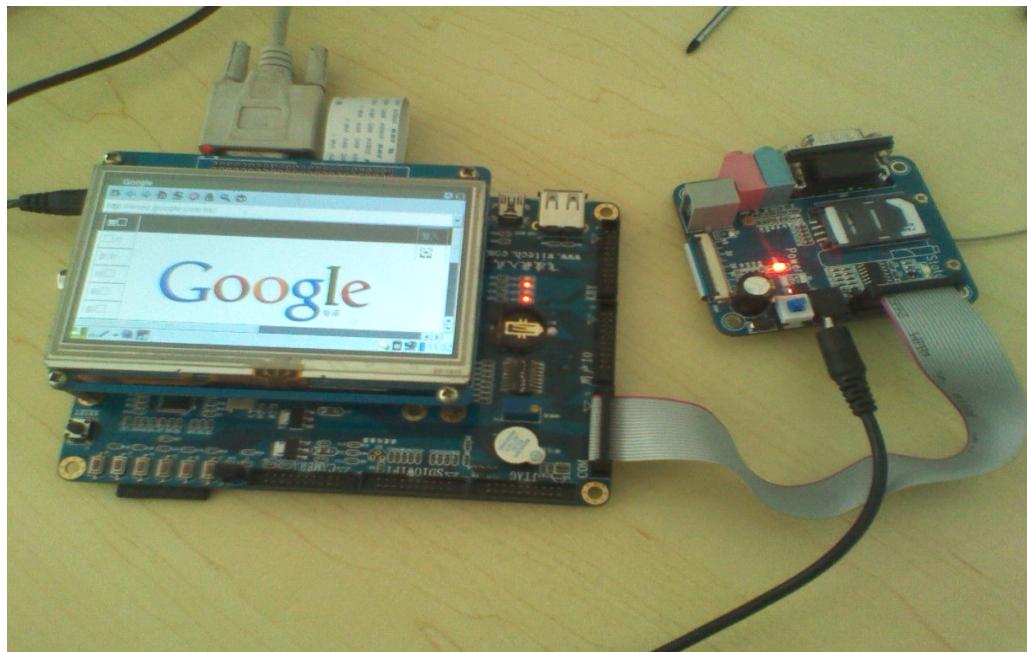
运行 `can_client` 的开发板串口信息显示如下:

```
[root@FORLINX6410]# can_client
can0 can_ifindex = 2
Send a CAN frame from interface 2
```

up、`can_server`、`can_client` 源码路径: 基础光盘\Linux-3.0.1\apptest\can

5-9 GPRS模块测试

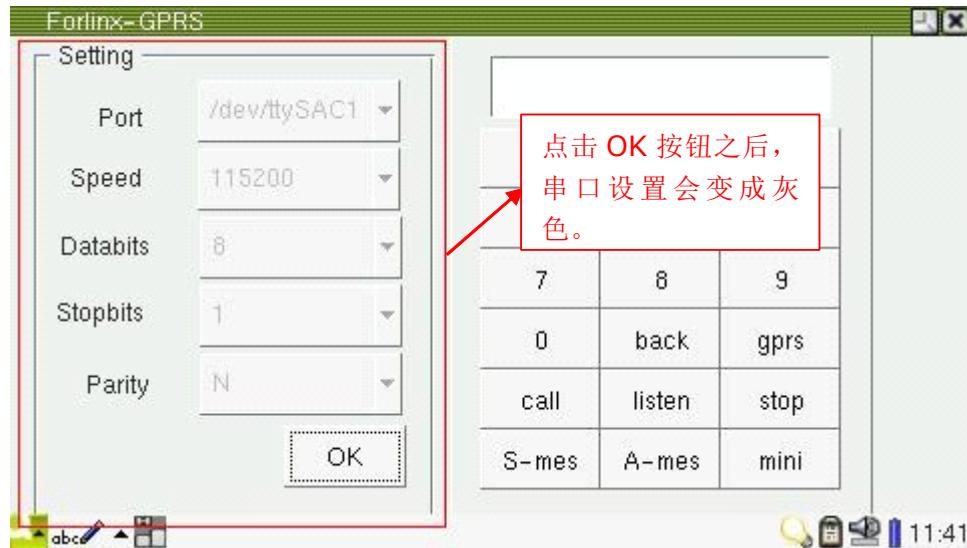
连接方法: 将插有手机卡的飞凌 GPRS 模块通过硬排线直接连到 OK6410 开发板 20pin 的 COM 口上, 连接方式如图。开发板和 GPRS 模块都接 5V 电源。本测使用的是神州行电话卡。



步骤 1.运行软件。在 qtopia 桌面中打开 ForlinxTest 选项标签，可以看到“gprs”图标，“gprs”是 GPRS 模块的应用程序，如图：



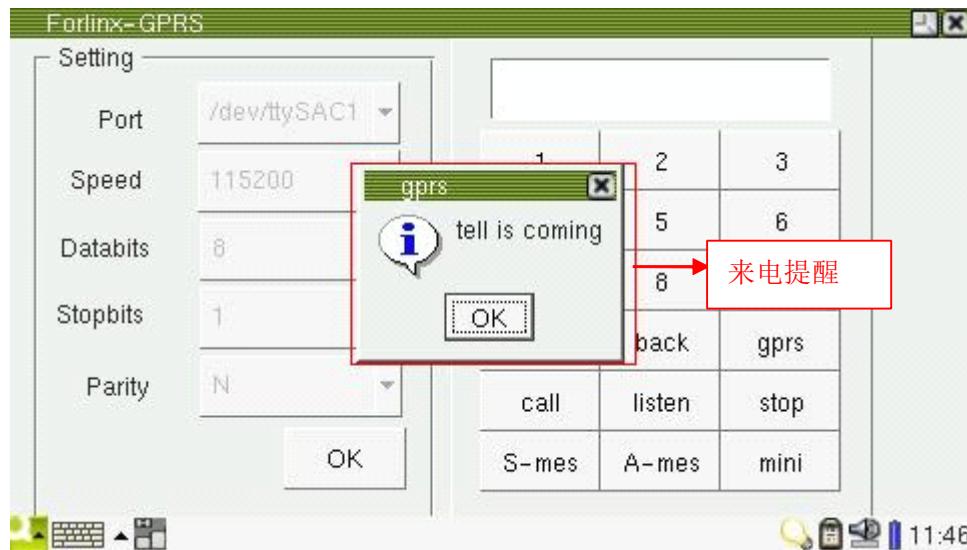
步骤 2.设置串口。在左边 Setting 里边设置串口信息，默认 gprs 用串口/dev/ttySAC1。点击 OK 即可设置成功。此时左边串口设置选项会变灰色。如果设置错误，请重启 gprs 软件。



步骤 3.打电话。设置好串口后，在电话号码栏里边输入对方电话，单击 call 按钮即可拨通电话。此时电话号码栏里显示“calling 电话号码”。

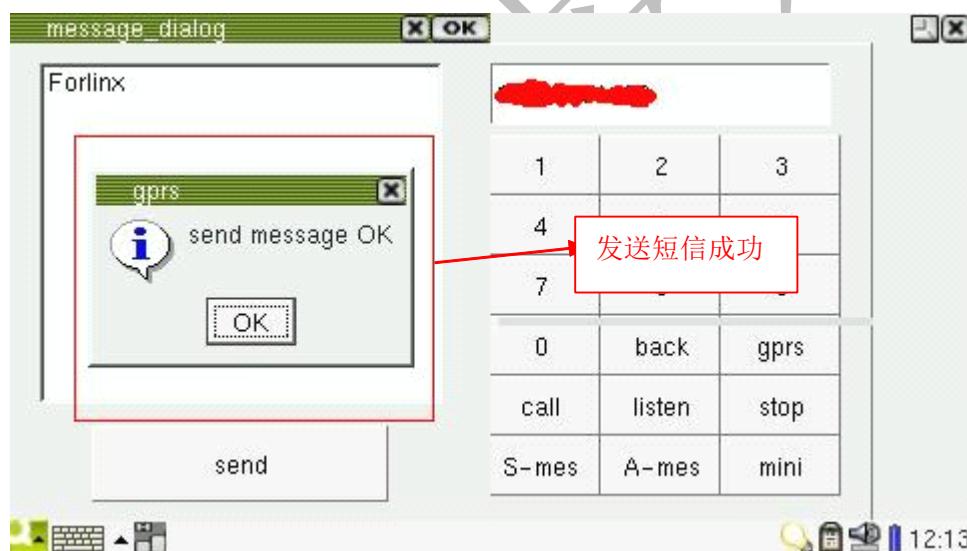


步骤 4.监听电话。如果有来电，会弹出来电提示。

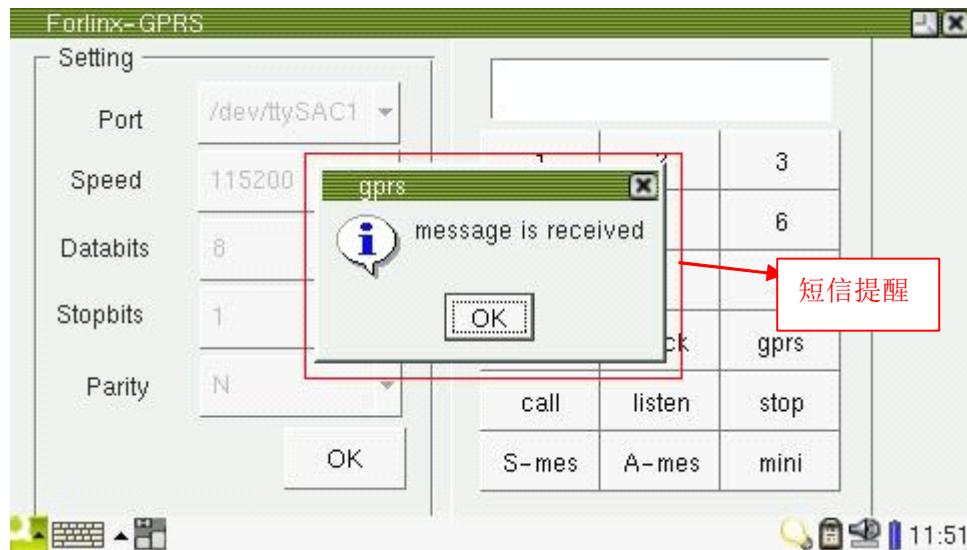


点击 listen 进行通话，点击 stop 挂断电话。

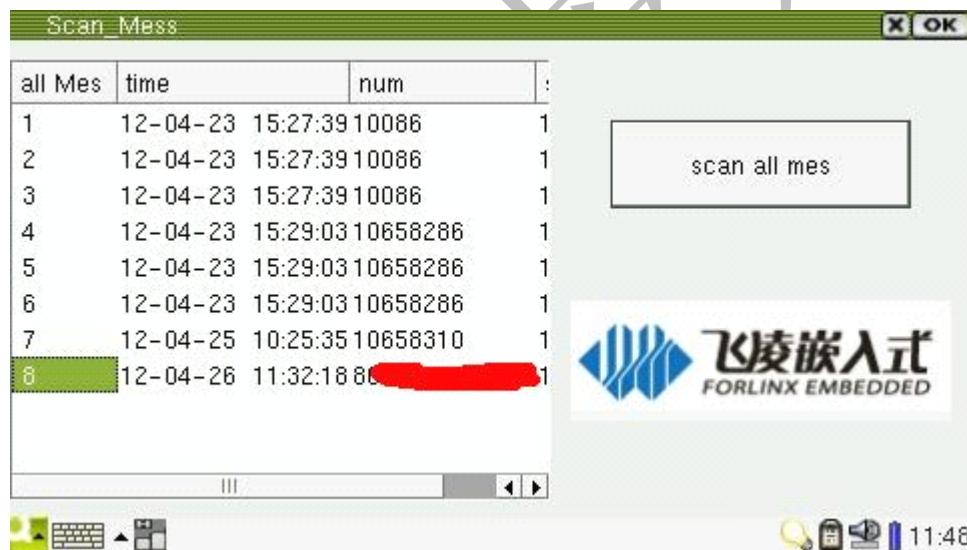
步骤 5.发送短信。回到主界面，在电话号码栏里边输入电话，单击 S-mes 按钮编辑短信。
点击 send 按钮即可发送短信。



步骤 6.监听短信。如果收到短信，会弹出短信提示。



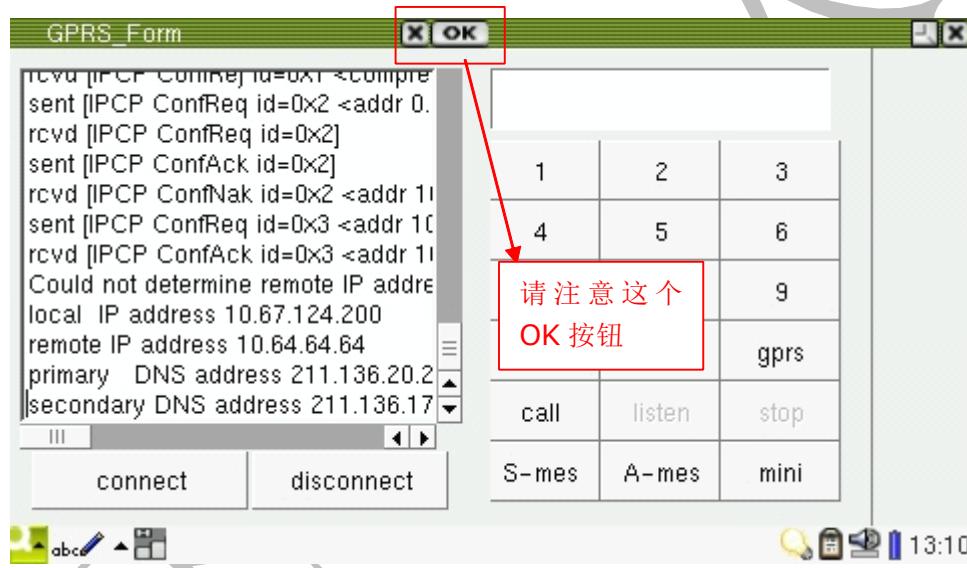
步骤 7.查看短信。单击 A-mes 按钮，弹出短信浏览对话框，再单击 scan all mes 按钮即可浏览所有短信。注意：在点 A-mes 按钮之前，保证电话号码栏为空，否则会进入编辑短信模式。



双击某个短信，即可显示短信内容。



步骤 8. 测试 gprs 上网功能。单击主界面上 gprs 按钮，弹出 gprs 对话框，单击 connect 即可拨号上网。



点击 gprs 界面右上角 ok，然后点击主界面 mini 按钮，将界面隐藏，启动浏览器。



注意：因为 GPRS 模块只有一个串口，打开 GPRS 功能之后，串口会被占用，所以打电话、发短信不能和 GPRS 功能同时使用。该软件暂不支持超长短信。

gprs 应用程序源码路径：

基础光盘\ Linux-3.0.1\apptest\QT ApplicationTest\gprs

第六章 在主机上搭建 Linux 开发环境

写在环境搭建的最前面：

开发环境是开发人员在开发过程当中，**所需的软硬件**。开发环境并不是一个固定的样式，在这里，我们详细讲解一个嵌入式 Linux 开发环境搭建的方法。您已经对嵌入式开发非常了解的话，可以按照自己的需求来搭建环境。如果和本手册环境不一样而产生报错，您可以从国内一些大 Linux 论坛和网站搜索相关的信息来解决。本册介绍的环境经过飞凌的测试，如果对嵌入式开发不是非常熟悉的朋友，希望您按照飞凌提供的方法来搭建环境。各位朋友可以放心按照本手册说明的方法来搭建开发环境。

本章所需文件路径：

文件名（文件用途）	文件在基础光盘中的路径
ubuntu-12.04-desktop-i386.iso (ubuntu12.04 安装映像)	实用工具\
FORLINX_linux-3.0.1.tar.gz (Linux-3.0.1 源码压缩包)	Linux-3.0.1\kernel_sourcecode\

6-1 安装Ubuntu12.04

Ubuntu 是一个以桌面应用为主的 Linux 操作系统。Ubuntu 拥有很多优点。相对于其他版本的 Linux，Ubuntu 也有着自己的优势。首先，**安装系统非常简单**，只需要非常少的设置即可，完全可以和 Windows 桌面系统媲美；其次，**图形界面很人性化**，模仿了在 xp 下常用的快捷键；还有，安装和升级程序时，可以通过网络，**由系统自行安装依赖的文件包**，从此不必再为 Linux 系统的依赖关系大伤脑筋。综合考虑大家的使用习惯和学习的需要，我们选用 Ubuntu Linux。

我们在产品光盘中提供了 Ubuntu12.04 光盘镜像 ‘ubuntu-12.04-desktop-i386.iso’，它位于基础资料光盘的“实用工具”目录下，以便于刻录成系统安装盘。

Linux 桌面系统版本众多，目前所有实验和源码在 ubuntu-12.04 版本测试可以通过。使用其他版本 Linux 桌面系统，可能会出现 gcc 编译器和库文件相关的问题。碰到类似问题，可以在 Linux 系统发行商的官方论坛上咨询和查询。如果对 Linux 不熟悉的用户，强烈建议使用飞凌介绍的方法。

Ubuntu官网：<http://www.ubuntu.org.cn>

可下载各个版本的 Ubuntu。当然，官网能找到各种关于 Ubuntu 的信息。

Ubuntu官方论坛：<http://forum.ubuntu.org.cn/>

可以找到大量的实用的 Ubuntu 资源，中文论坛。Ubuntu 也有官方的英文论坛。

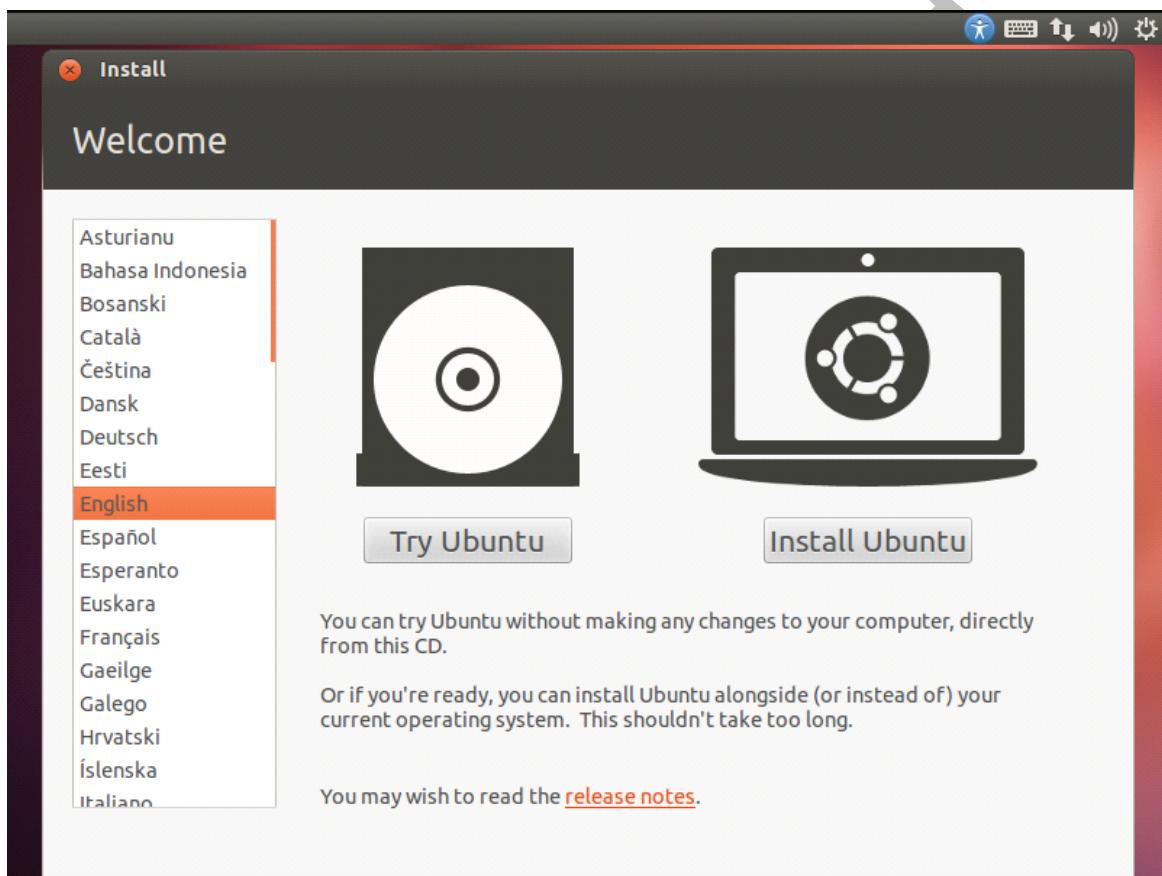
请注意：飞凌使用的Ubuntu12.04 是 32 位版本，考虑到大多用户的需求，我们选择了 32 位系统，未选择 64 位的Ubuntu12.04.

另外，为了满足各位用户的需求，在本手册的附录中，特对VMware虚拟机使用方法进行了详细的讲解。

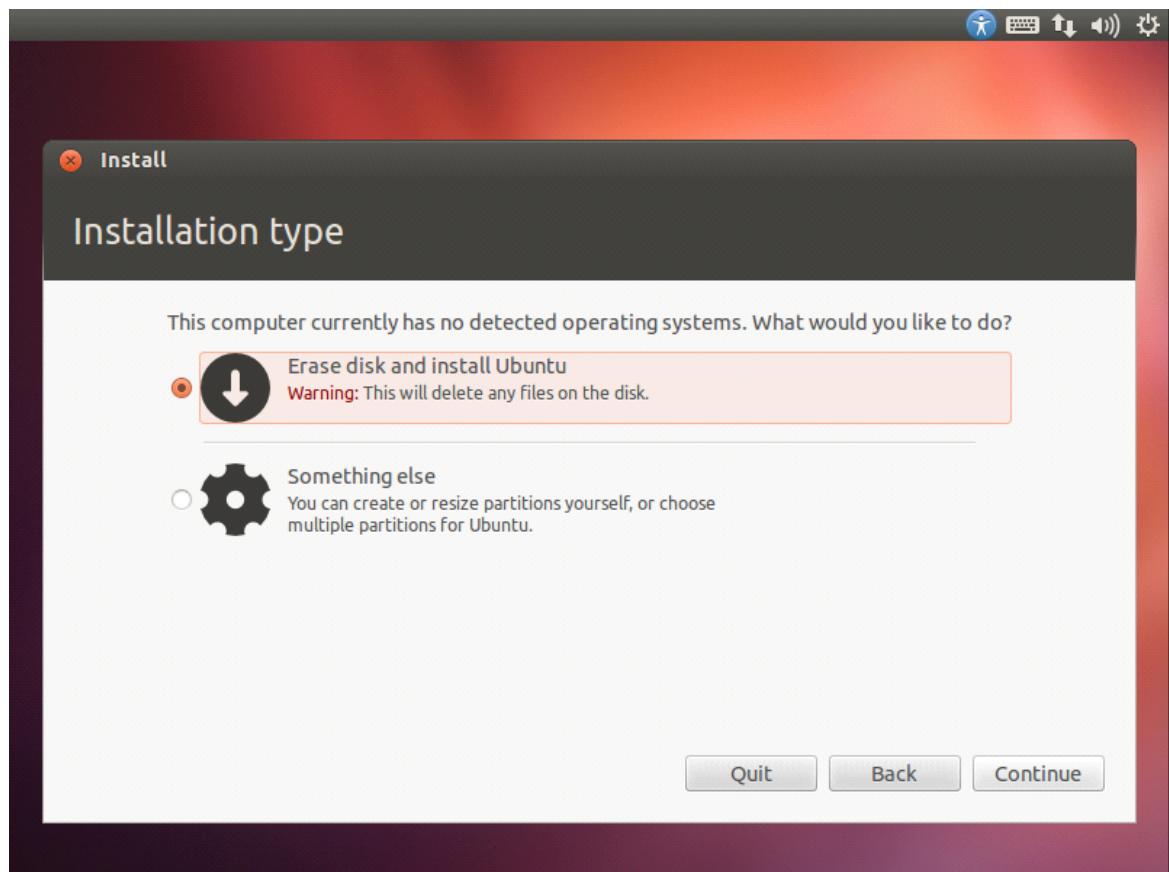
下面开始安装：

步骤 1. 首先准备一张Ubuntu12.04 的安装光盘。将光盘插入光驱，在 PC 的 bios 中把 PC 启动方式设置为光驱启动，启动 PC。

步骤 2. 启动 PC 后，安装盘会提示选择安装语言种类。使用 PC 键盘的方向键选择在安装过程中显示的语言，在这里我们选择 English.

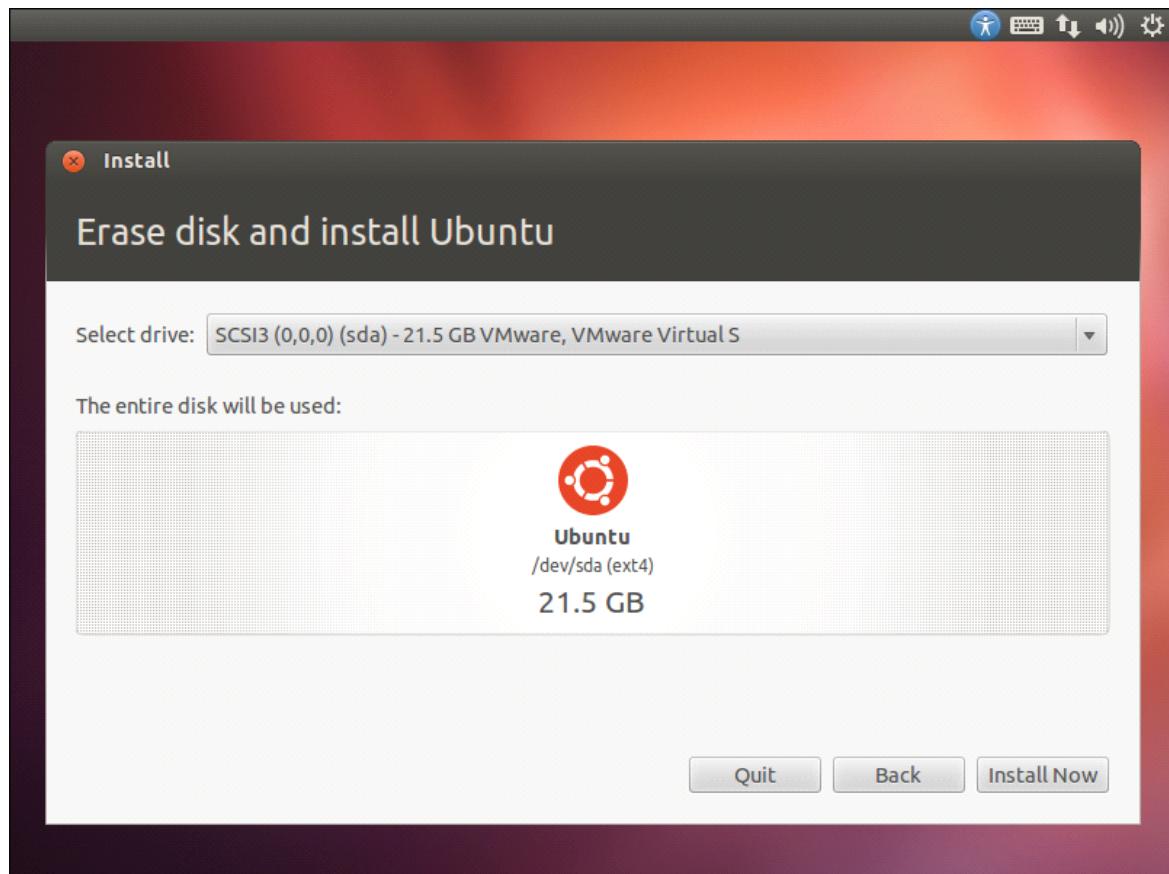


选择“Install Ubuntu”。

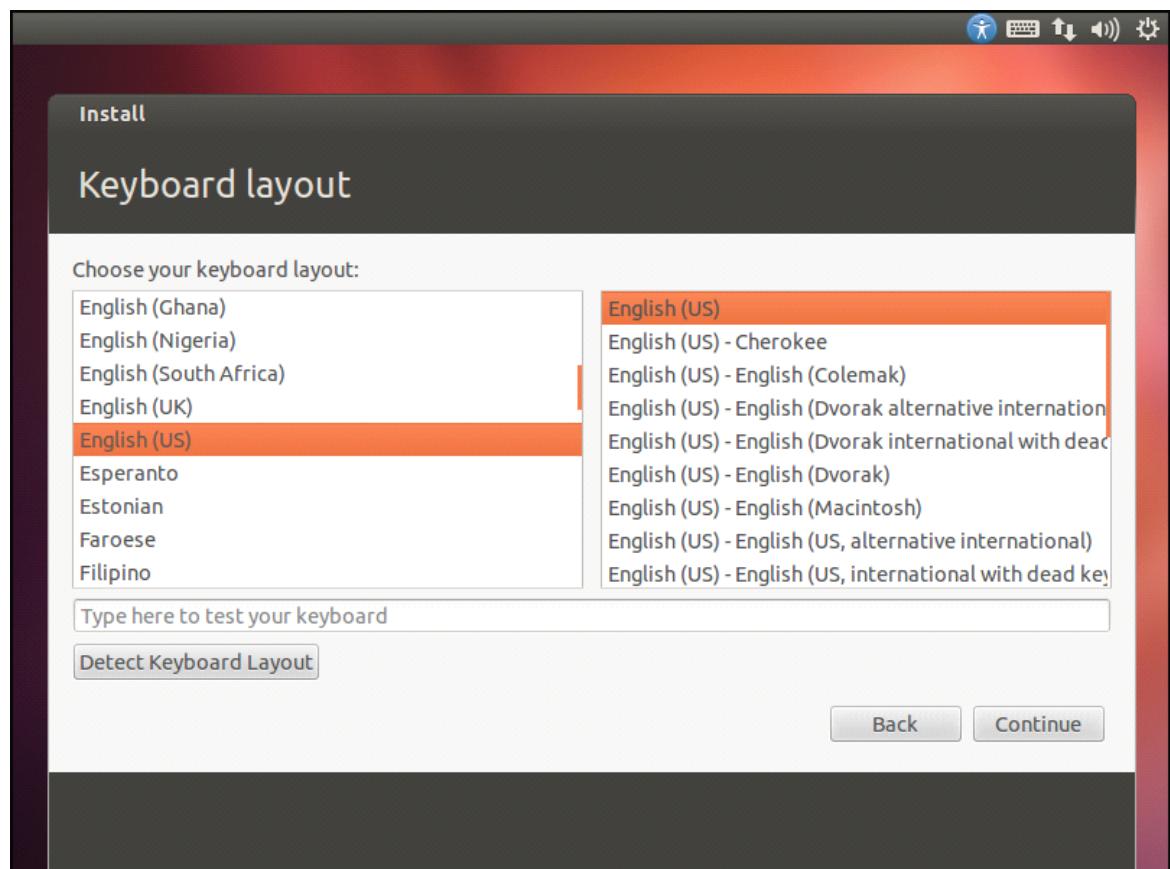


Continue:

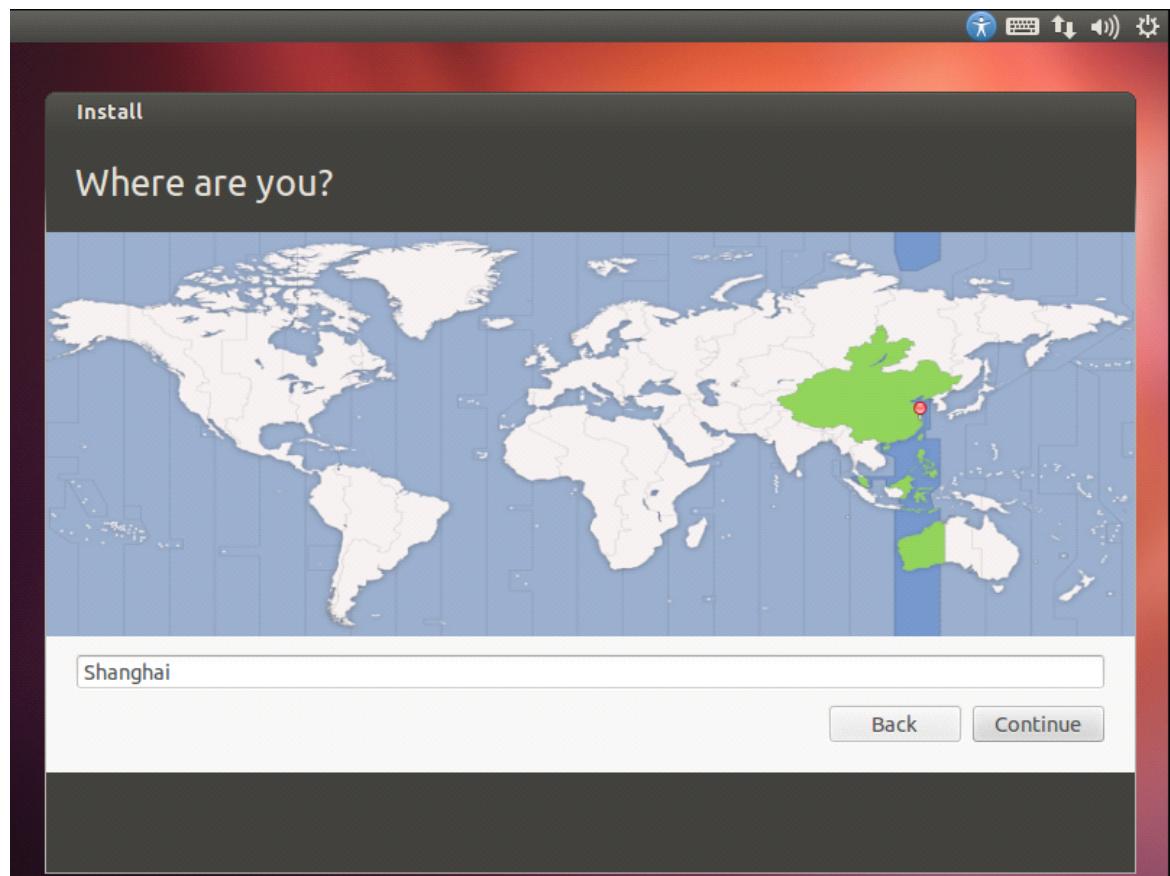




硬盘空间以及挂载点的分配。这里选择默认。也可根据个人需要来进行设置。
单击 **Install Now**

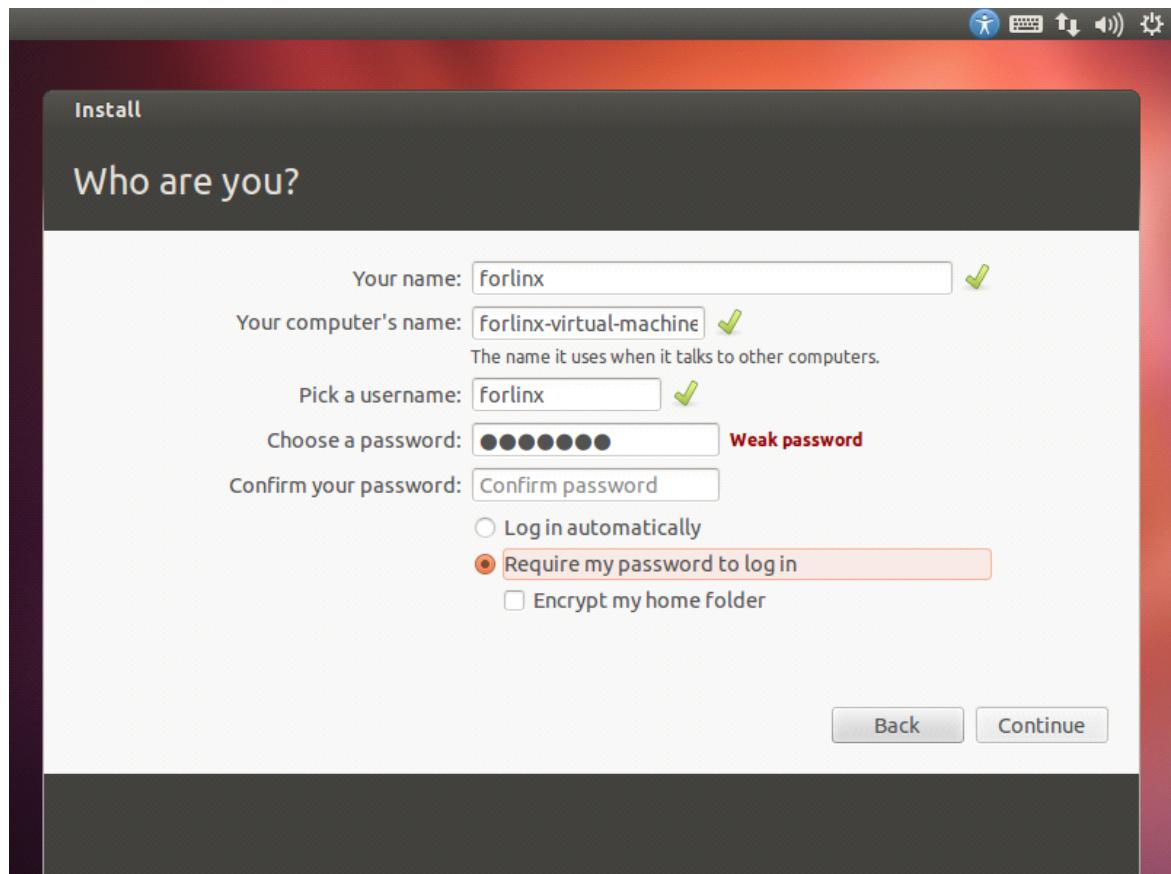


选择键盘布局， 默认即可， 单击 Continue。



选择所在地,上海, Continue:





输入系统用户名和密码，这里输入的用户名:forlinx，密码:123456。

注意：这里的用户名属于普通用户，不具备 root 用户权限，系统如何以 root 登陆，下一节有详细说明。

继续 Continue, 系统会完成安装。

6-2 将Ubuntu设置为root用户自动登录

嵌入式交叉编译，经常需要 root 用户的权限，ubuntu12.04 默认是不允许 root 登录的，在登录窗口只能看到普通用户和访客登录。以普通身份登陆 Ubuntu 后我们需要做一些修改，用于支持 root 用户登陆。

普通用户登录后，需要在终端窗口中执行命令切换到超级用户模式，在 Ubuntu12.04 如何进入终端窗口，请参看 [6-4-1 Linux 终端](#)一节关于终端使用介绍。

在终端窗口里面输入: `sudo -s` 回车 ,然后输入安装 Ubuntu12.04 时设置的系统密码，即可进入 root 用户权限模式.在终端窗口中执行 `gedit /etc/lightdm/lightdm.conf`.

增加 `greeter-show-manual-login=true allow-guest=false` 两行.

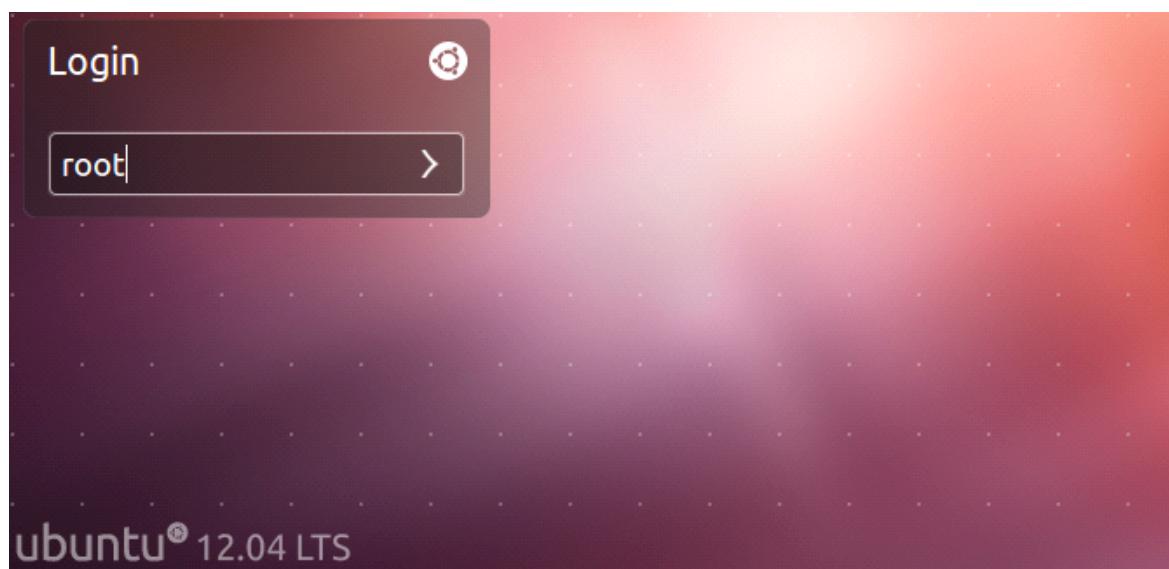
修改完的整个配置文件如下:

```
[SeatDefaults]
greeter-session=unity-greeter
user-session=ubuntu
greeter-show-manual-login=true #手工输入登陆系统的用户名和密码
allow-guest=false #不允许 guest 登录
```

然后我们启动 root 帐号: 在终端窗口中执行 `passwd root` 命令, 根据提示输入 root 帐号的密码。

重启 ubuntu，登录窗口会有“登录”选项，这时候我们就可以通过 root 登录了。

如下图所示:



6-3 设置Ubuntu网络参数

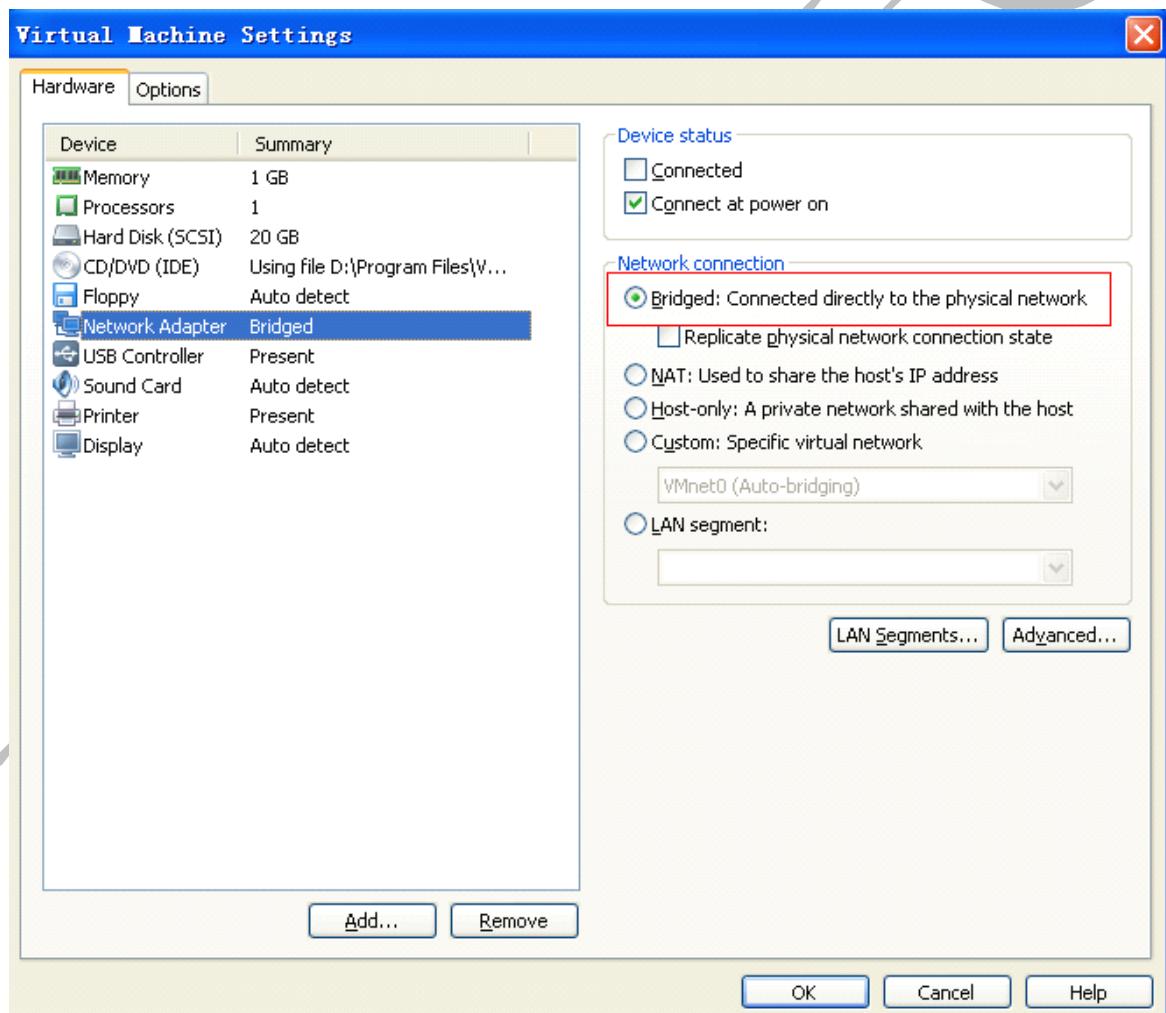
因为每个 PC 的网络环境都不一定一样，所以需要您根据自己的实际情况来设置 Ubuntu 的网络，如果设置不成功，可以去 Ubuntu 的官方论坛上咨询。

接下来，为您提供一种网络的设置方法，仅供参考。

步骤 1： 虚拟机使用固定 IP 地址的网络方式

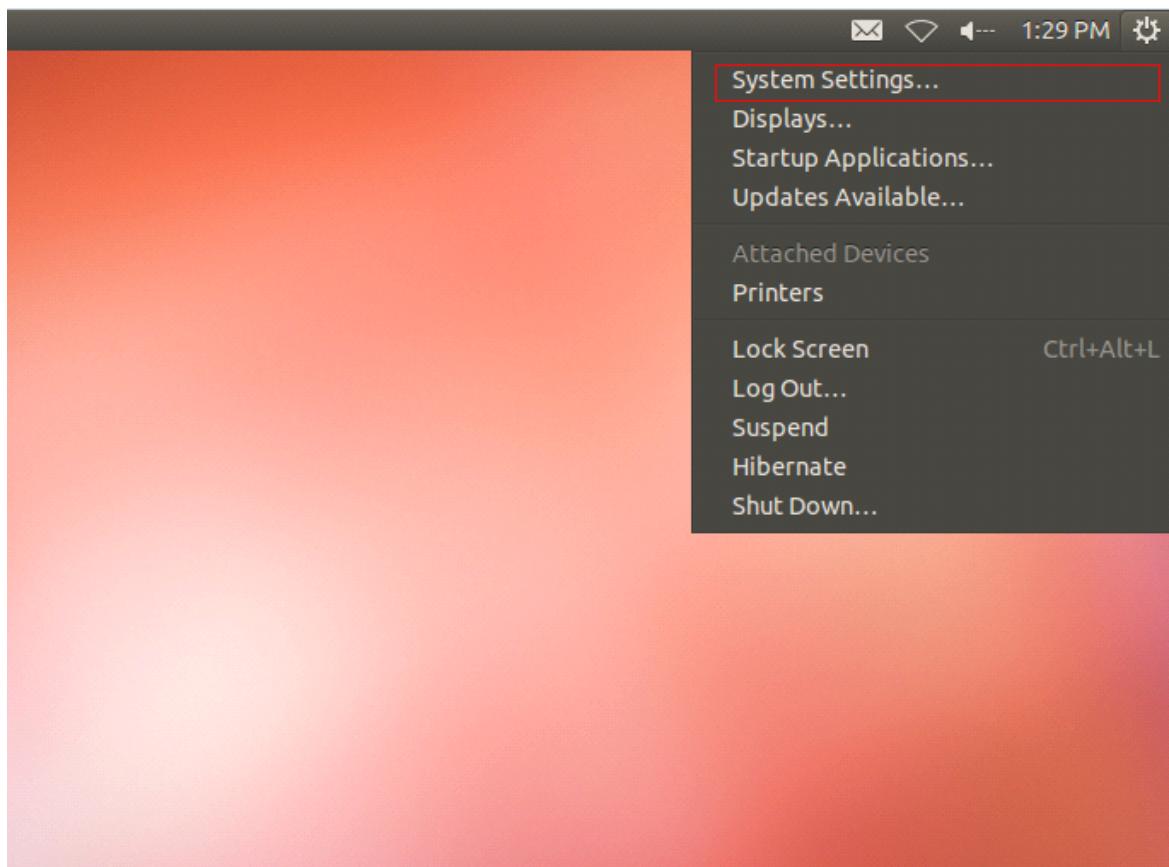
默认情况下，虚拟机安装完成后与宿主机共享一个 IP 地址，没有属于自己的 IP，这里首先需要设置虚拟机的网络联系方式为桥接方式。

单击 VM 菜单下面的 **Settings** 项，弹出虚拟机设置对话框，如下图：



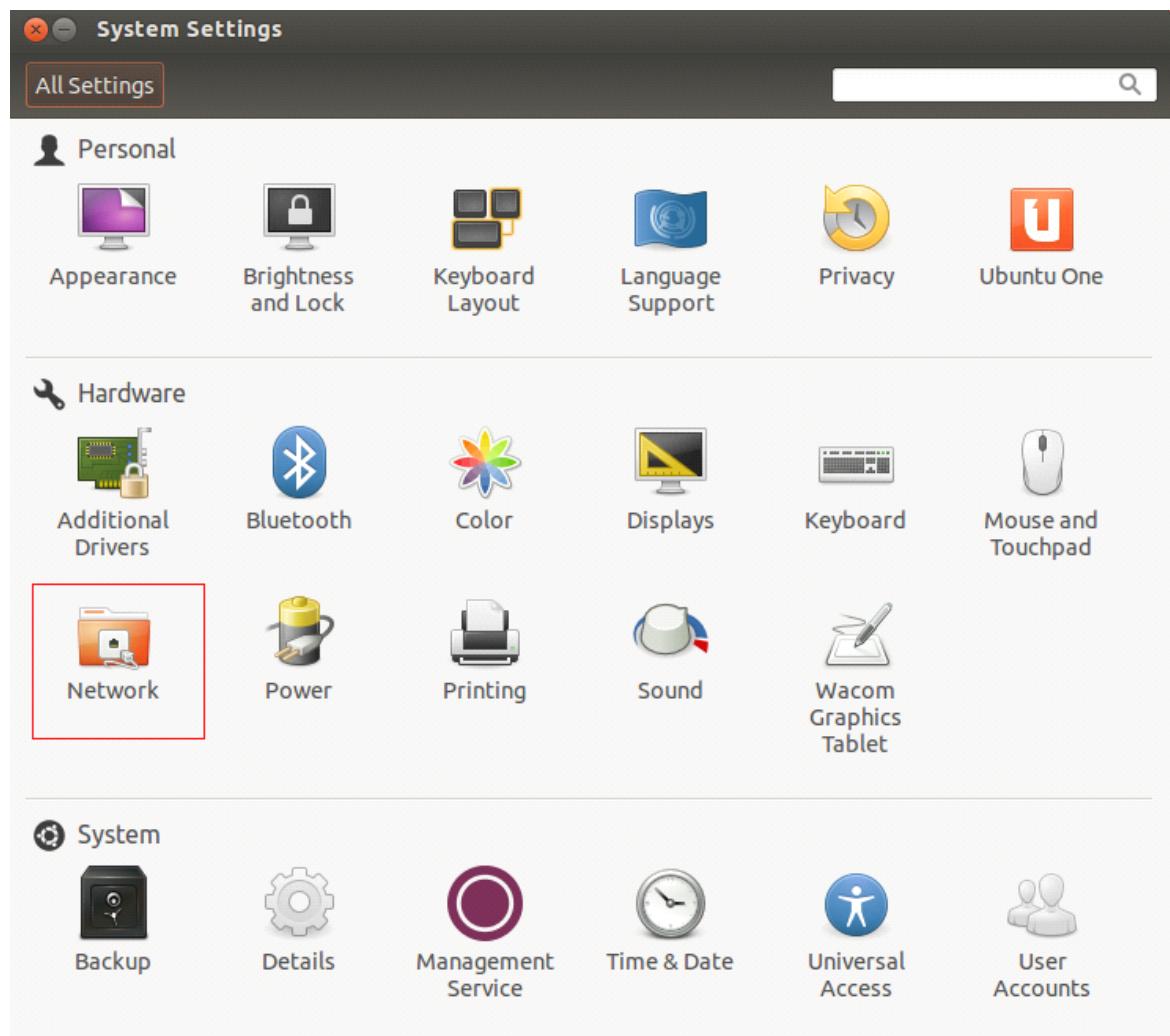
选择 **Bridged** 连接，确定即可。

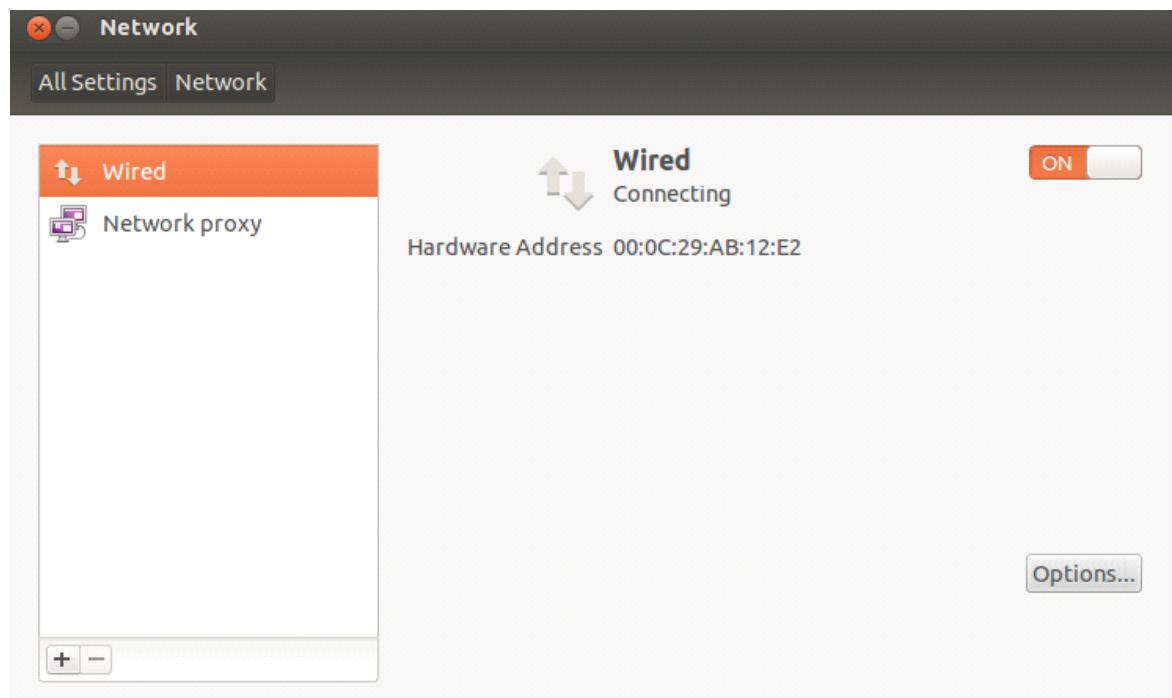
步骤 2 . 启动 Ubuntu，root 用户登陆系统，单击桌面最右上端的按钮，弹出如下选项：



选择 System Settings，进入如下设置：

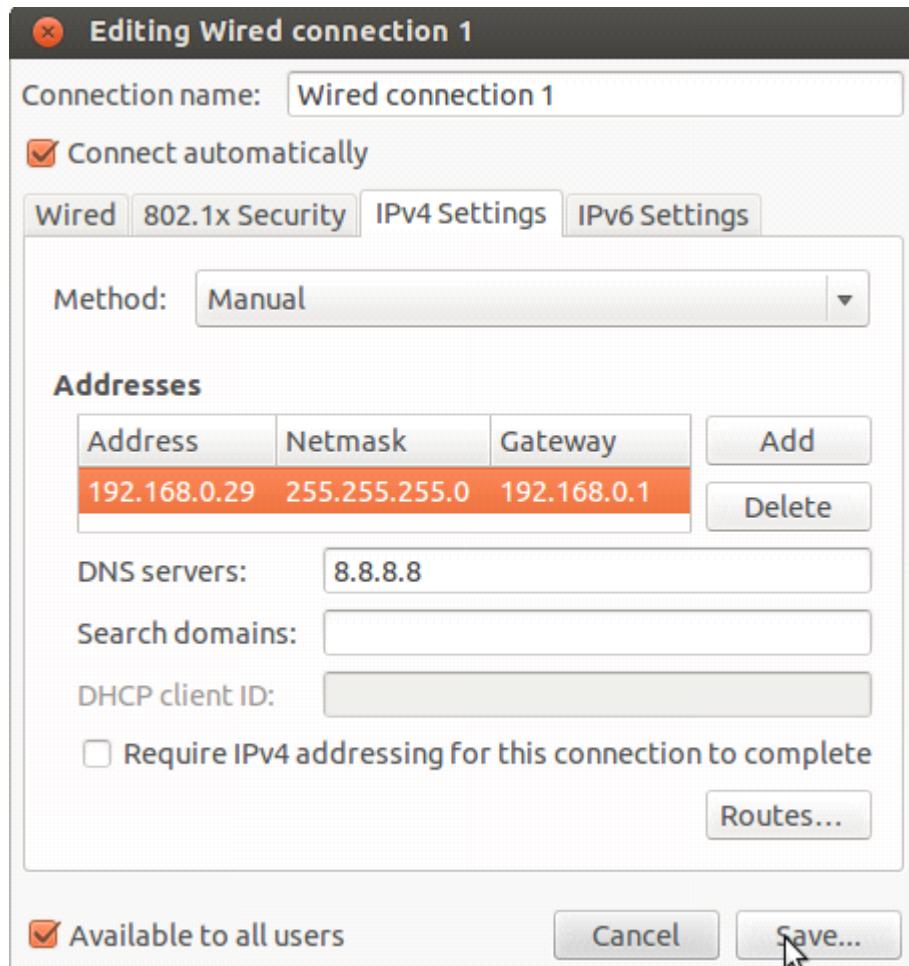






单击 Options 按钮:





选择 **IPV4** 设置，输入您的 IP 地址，子网掩码，网关，DNS，保存，网络设置成功。

测试一下，笔者的宿主机 IP 为 192.168.0.30，用虚拟机来 ping 宿主机：

```
root@forlinx-virtual-machine:~# ping 192.168.0.30
PING 192.168.0.30 (192.168.0.30) 56(84) bytes of data.
64 bytes from 192.168.0.30: icmp_req=1 ttl=64 time=0.785 ms
64 bytes from 192.168.0.30: icmp_req=2 ttl=64 time=0.239 ms
64 bytes from 192.168.0.30: icmp_req=3 ttl=64 time=0.174 ms
64 bytes from 192.168.0.30: icmp_req=4 ttl=64 time=0.179 ms
64 bytes from 192.168.0.30: icmp_req=5 ttl=64 time=0.251 ms
64 bytes from 192.168.0.30: icmp_req=6 ttl=64 time=0.115 ms
64 bytes from 192.168.0.30: icmp_req=7 ttl=64 time=0.249 ms
```

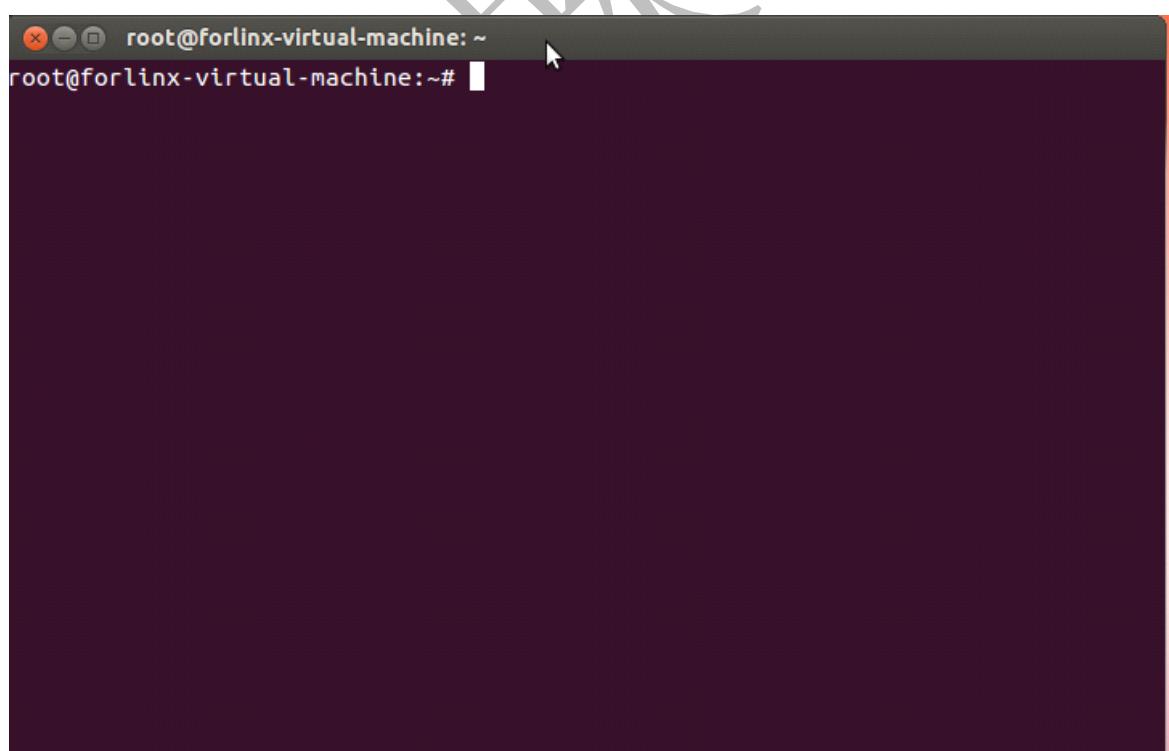
6-4 Ubuntu常用应用程序

6-4-1 Linux终端

在 Linux 系统中终端是一个很实用的与操作系统交互的窗口，您可以使用终端来编译应用程序，启动各种系统服务，在 Linux 系统中占据着非常重要的位置。



以后每次单击启动面板上的图标即可运行终端窗口：



6-5 安装交叉编译器

将 arm-linux-gcc-4.3.2.tgz 文件拷贝到 Ubuntu 的 /forlinx 目录下，该文件位于用户基础资料光盘的“实用工具”文件夹中。在 Ubuntu 中新建一个终端，输入下面的命令安装交叉编译器：

```
#cd /forlinx      (进入/forlinx 目录)
#mkdir /usr/local/arm    (创建目录, 若目录已存在会提示错误, 跳过即可)
#tar zxvf arm-linux-gcc-4.3.2.tgz -C /
编译器解压到/usr/local/arm
```

把交叉编译器路径添加到系统环境变量中，以后可以直接在终端窗口中输入 arm-linux-gcc 命令来编译程序。

在终端中执行：

```
gedit /etc/profile
```

添加以下四行到该文件中：

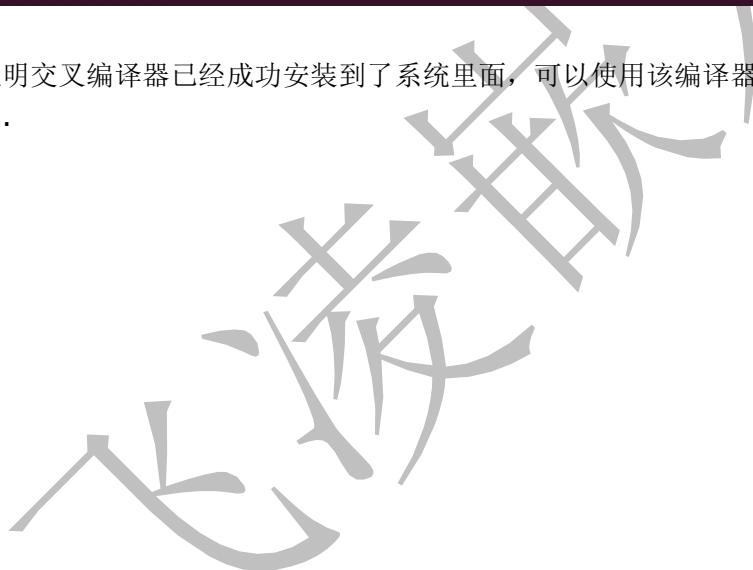
```
export PATH=/usr/local/arm/4.3.2/bin:$PATH
export TOOLCHAIN=/usr/local/arm/4.3.2
export TB_CC_PREFIX=arm-linux-
export PKG_CONFIG_PREFIX=$TOOLCHAIN/arm-none-linux-gnueabi
```

保存，退出。

重新启动系统，在终端里面执行 arm-linux-gcc 回车：

```
root@forlinx-virtual-machine: ~
root@forlinx-virtual-machine:~# arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure --build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers=gnu --enable-_cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72' --with-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --with-build-sysroot=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/libc --with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
root@forlinx-virtual-machine:~#
```

说明交叉编译器已经成功安装到了系统里面，可以使用该编译器来编译 Uboot 代码和内核代码了。



第七章 编译 UBOOT 和 Linux 内核

本章说明了 U-boot 和 Linux 内核在 PC Linux 的编译方法。在进行本章实验之前，请参考第五章搭建好开发环境。

请注意：在本章中，如没有特殊说明，所执行命令以及设置的环境均为 PC 机的 Linux。在每条命令开头加符号 ‘#’ 以表明命令的开始。

文件名（文件用途）	文件在基础光盘中的路径
uboot1.1.6_FORLNX_6410.tgz (uboot 源码压缩包)	Linux-3.0.1\uboot_sourcedode\
FORLNX_linux-3.0.1.tar.gz (Linux-3.0.1 源码压缩包)	Linux-3.0.1\kernel_sourcecode\

7-1 编译 u-boot-1.1.6

将 uboot 源码压缩包 ‘uboot1.1.6_FORLINX_6410.tgz’ 拷贝到 Ubuntu 的 `/forlinx` 目录下，解压缩并编译，Ubuntu 下操作过程如下所示：

```
#tar -zxf uboot1.1.6_FORLINX_6410.tgz (解压缩 uboot 源码, 如图)
```



注意：飞凌开发板目前有 128M 内存和 256M 内存两个硬件版本，Uboot1.1.6 一套代码适用于飞凌两个类型的开发板，在配置 Uboot 的时候需要指明是 128M 版本还是 256M 版本。

7-1-1 编译 128M 内存开发板 Uboot 方法:

进入 uboot1.1.6 目录、配置 config、编译：

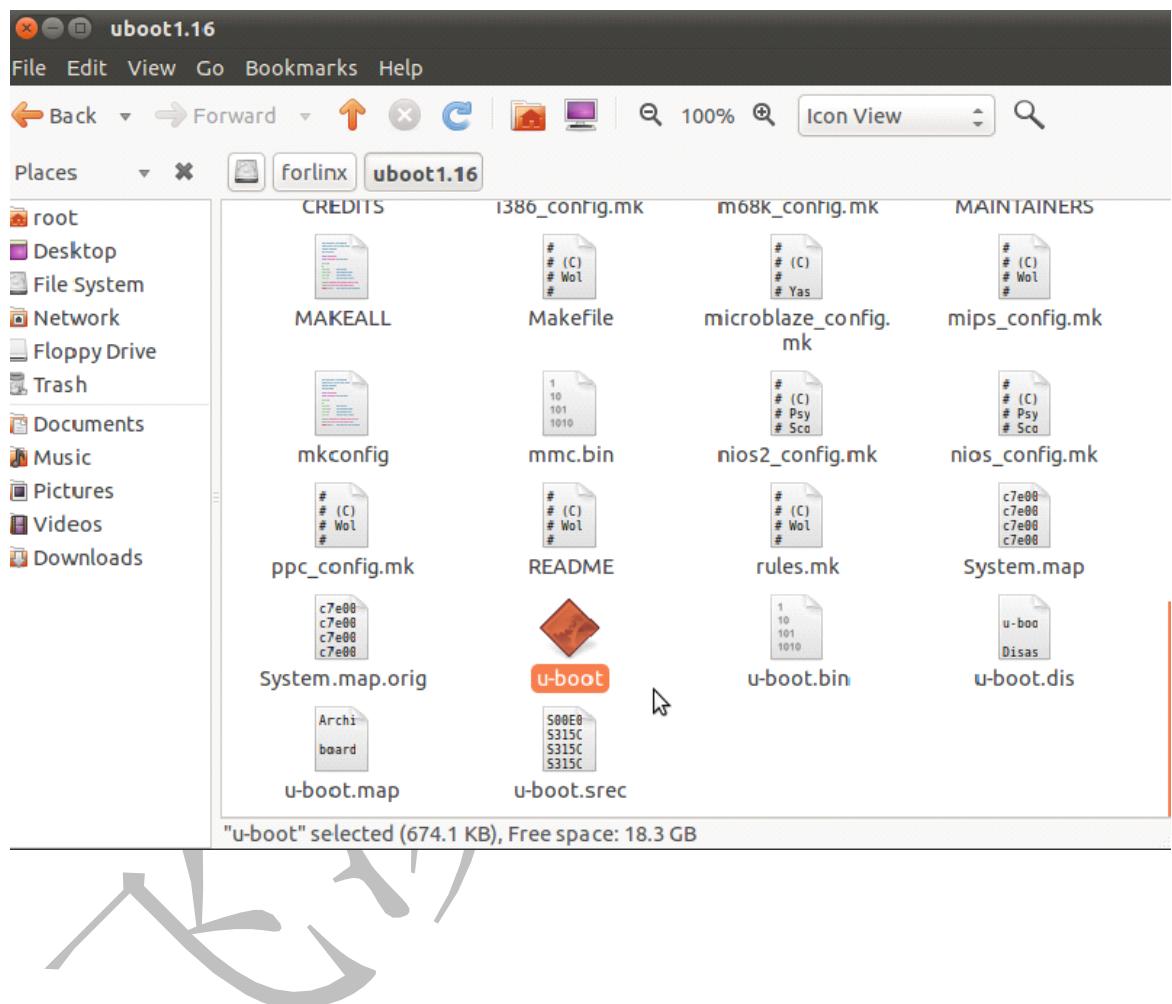
#cd uboot1.1.6 (进入 uboot 源码的目录)

make forlinx_nand_ram128_config (配置适用于 128M 内存开发板的 config)

#make clean (删除以前编译的文件)

#make (编译)

如果编译成功，将在 ‘uboot1.1.6’ 目录下产生名为 ‘u-boot.bin’ 的二进制文件。该文件即我们需要烧写到 Nandflash 的 U-boot 映像文件。如下图所示：



7-1-2 编译 256M 内存开发板 Uboot 方法:

进入 uboot1.1.6 目录、配置 config、编译:

#cd uboot1.1.6 (进入 uboot 源码的目录)

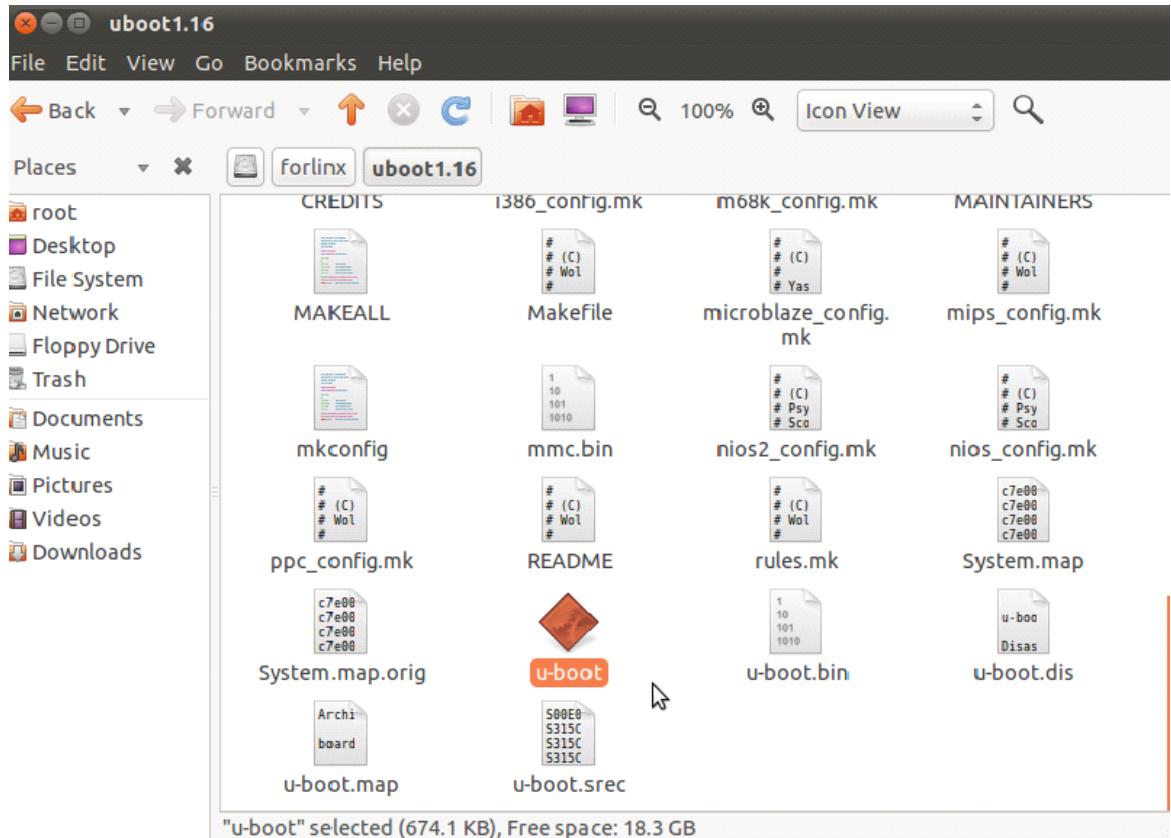
make forlinx_nand_ram256_config (配置适用于 256M 内存开发板的 config)

```
root@forlinx:/forlinx/uboot1.1.6
File Edit View Search Terminal Help
drivers/sk98lin/libsk98lin.a post/libpost.a post/cpu/libcpu.a common/libcommon.a
|sed -n -e 's/.*\(\_u_boot_cmd_\.*\)/-\u\1/p'|sort|uniq`;\
cd /forlinx/uboot1.1.6 && /usr/local/arm/4.3.2/bin/arm-linux-ld -Bstatic -T /forlinx/uboot1.1.6/board/samsung/smdk6410/u-boot.lds -Ttext 0xC7E0000
0 $UNDEF_SYM cpu/s3c64xx/start.o \
--start-group lib_generic/libgeneric.a board/samsung/smdk6410/libsmdk6410.a
cpu/s3c64xx/libs3c64xx.a cpu/s3c64xx/s3c6410/libss3c6410.a
lib_arm/libarm.a fs/cramfs/libcramfs.a fs/fat/libfat.a fs/fdos/libfdos.a fs/jffs2/
libjffs2.a fs/reiserfs/libreiserfs.a fs/ext2/libext2fs.a net/libnet.a disk/libdisk.a
rtc/librtc.a dtt/libdtt.a drivers/libdrivers.a drivers/nand/libnand.a
drives/nand_legacy/libnand_legacy.a drivers/onenand/libonenand.a
drivers/sk98lin/libsk98lin.a post/libpost.a post/cpu/libcpu.a common/libcommon.a --end-group -L /usr/local/arm/4.3.2/bin/.../lib/gcc/arm-none-linux-gnueabi/4.3.2/armv4t -lgcc \
-Map u-boot.map -o u-boot
/usr/local/arm/4.3.2/bin/arm-linux-objcopy --gap-fill=0xff -O srec u-boot u-boot.srec
/usr/local/arm/4.3.2/bin/arm-linux-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
/usr/local/arm/4.3.2/bin/arm-linux-objdump -d u-boot > u-boot.dis
root@forlinx:/forlinx/uboot1.1.6# make forlinx_nand_ram256_config
Configuring for smdk6410 board which boot from NAND ram256...
root@forlinx:/forlinx/uboot1.1.6#
```

#make clean (删除以前编译的文件)

#make (编译)

如果编译成功，将在 ‘uboot1.1.6’ 目录下产生名为 ‘u-boot.bin’ 的二进制文件。该文件即我们需要烧写到 Nandflash 的 U-boot 映像文件，如下图所示：



7-2 编译 Linux-3.0.1

将压缩包 ‘FORLINX_linux-3.0.1.tar.gz’ 拷贝到你的工作目录下，解压缩：

```
#tar zxf FORLINX_linux-3.0.1.tar.gz
```

7-2-1 配置内核

您可能需要安装 ‘libncurses5’，以方便使用 ‘make menuconfig’ 命令，可以采用以下命令行来安装（这个步骤需要 pc 可以连接互联网）：

```
#sudo apt-get install libncurses5-dev
```

如果执行命令后无法找到 libncurses5-dev 安装包，那就需要先执行 `#sudo apt-get update`，再执行 `#sudo apt-get install libncurses5-dev`

7-2-2 编译内核

命令如下：

```
#make zImage
```

编译结束后将在内核源码目录的 arch/arm/boot 中得到 Linux 内核映像文件：zImage

7-2-3 开发板驱动源码路径

- (1)DM9000 网卡驱动
drivers/net/dm9000.c
- (2)串口(包括三个串口驱动 0,1,2,4, 对应设备名/dev/ttySAC0,1,2,4)
drivers/tty/serial/s3c6400.c
- (4)LED 驱动
drivers/char/s3c6410_leds.c
- (5)看门狗驱动
drivers/watchdog/s3c2410_wdt.c
- (6)触摸屏驱动
drivers/input/touchscreen/s3c-ts.c
- (7)yaffs2 文件系统源代码目录
fs/yaffs2
- (9)SD/MMC 卡驱动源代码目录
drivers/mmc
- (10)Nandflash 驱动
drivers/mtd/nand
- (11)WM9714 音频驱动目录
sound/
- (12)LCD 驱动
drivers/video/samsung
- (13)优盘支持驱动
drivers/usb/storage
- (15) USB 鼠标、键盘源代码
drivers/hid
- (16)CMOS 摄像头驱动
drivers/media/video/samsung/fimc
- (17)USB 无线网卡驱动(适用于 rt2501 usb wifi)
drivers/net/wireless/rt2x00
- (18)各种 USB 转串口驱动
drivers/usb/serial/
- (19)SDIO WIFI 无线网卡驱动
drivers/net/wireless/libertas
- (20)CPU 自带 TV 输出驱动
drivers/media/video/samsung/tv
- (21)飞凌 6410 专用矩阵键盘硬件控制器驱动
drivers/input/keyboard
- (22)ADC 驱动
drivers/char/forlinx6410_adc.c
- (23)PWM 蜂鸣器驱动
drivers/char/forlinx6410_pwm.c

- (24)HS0038B 红外驱动
drivers/char/forlinx6410_irda.c
- (25) 温度传感器 DS18B20
drivers/char/forlinx6410_18b20.c



第八章 制作 yaffs2 文件系统映像

8-1 准备好文件系统

这里我们以飞凌提供的文件系统为例，为您演示如何制作 yaffs2 文件系统映像。

FileSystem-Yaffs2.tar.gz 是我们提供的文件系统目录，用户可以使用此目录制作 Yaffs2 文件系统，且 FileSystem-Yaffs2.tar.gz 目录也用于 NFS 网络根文件系统挂载，NFS 挂载具体参考 [4-7-7 节 NFS 网络文件系统挂载](#)。

注意：本次发布的文件系统 FileSystem-Yaffs2.tar.gz 相对于上次文件系统添加了一些有用的 qtopia2.2.0 的测试程序，同时也删减了一些 qt4.7.1 的演示程序。

8-2 制作映像

在 基础光盘\Linux-3.0.1\filesystem\Yaffs2 文件系统制作工具 中有两个制作工具：
mkyaffs2image-nand2g 和 mkyaffs2image-nand256m

(1)

mkyaffs2image-nand256m 制作出的映像，适用于 **256M 字节 nandflash** 的开发板

制作命令：

```
#./mkyaffs2image-nand256m FileSystem-Yaffs2 rootfs.yaffs2
```

(2)

mkyaffs2image-nand2g 制作出的映像，适用于 **1G 或 2G 或者 4G 字节 nandflash** 的开发板

制作命令：

```
#./mkyaffs2image-nand2g FileSystem-Yaffs2 rootfs.yaffs2
```

最后生成 rootfs.yaffs2 是可以下载到开发板 nandflash 中的 yaffs2 文件系统映像。

注意：mkyaffs2image 可执行文件是使用 Linux3.0 源代码目录 yaffs2 文件夹下 utils 目录中的 mkyaffs2image.c 文件编译出来的，如果您有兴趣可以自己制作适合 256MB NandFlash 和 1G NandFlash 的 Yaffs2 工具。(2G/4G nandflash 同 1G 芯片制作方式一样)

第九章 多媒体硬件编解码测试

9-1 多媒体功能简介

S3C6410 内部集成的多媒体编解码器(MFC)支持 MPEG4/H.263/H.264 的编码与解码，并支持 VC1 解码，解码性能可达 720x480 30fps 或者 720x576 25fps

我们常见的 H.264 编码的 AVI 文件和 MPEG4 编码的 MP4 文件可以在 s3c6410 芯片上流畅播放，这样的音视频文件分辨率通常要小于等于 720*480，如果大于这个尺寸，需要使用转换工具转换成 6410 支持的格式，具体转换方法我们会在下面的章节中说明。

另外多媒体测试程序也可以测试飞凌的 ov9650 摄像头的预览功能，视频数据 H.264 编码功能，YUV420 和 YUV422 视频数据与 RGB 数据转换等功能，是您学习视频编解码的第一手资料。

我们提供的 Linux3.0 内核源码已经包含了 6410 多媒体硬件编解码驱动和 ov9650 摄像头驱动，并且完全开源，我们把它放到了 Linux3.0 内核中，方便用户学习和产品开发。

9-2 多媒体综合测试一

三星提供的多媒体测试程序 multimediatest，可以测试 MPEG4, H.264,H.263 的解码测试和 H.264 的编码测试，JPEG 的编解码测试，我们在三星提供的版本基础上进行了修改，可以支持飞凌 3.5 寸，4.3 寸，5.6 寸，7 寸，8 寸，10.4 寸，VGA，HDMI 输出显示，源代码会在光盘中提供，您可以拿来学习和修改。

9-2-1 编译 MultimediaTest 程序

飞凌提供的多媒体硬件编解码的综合测试程序 Multimedia_DD-2012-01-07.tar.gz，位于开发板基础资料光盘的“Linux3.0\apptest\Multimedia”目录下。请将其拷贝到 Ubuntu 的/forlinx 目录下，解压，得到 Multimedia_DD 目录。

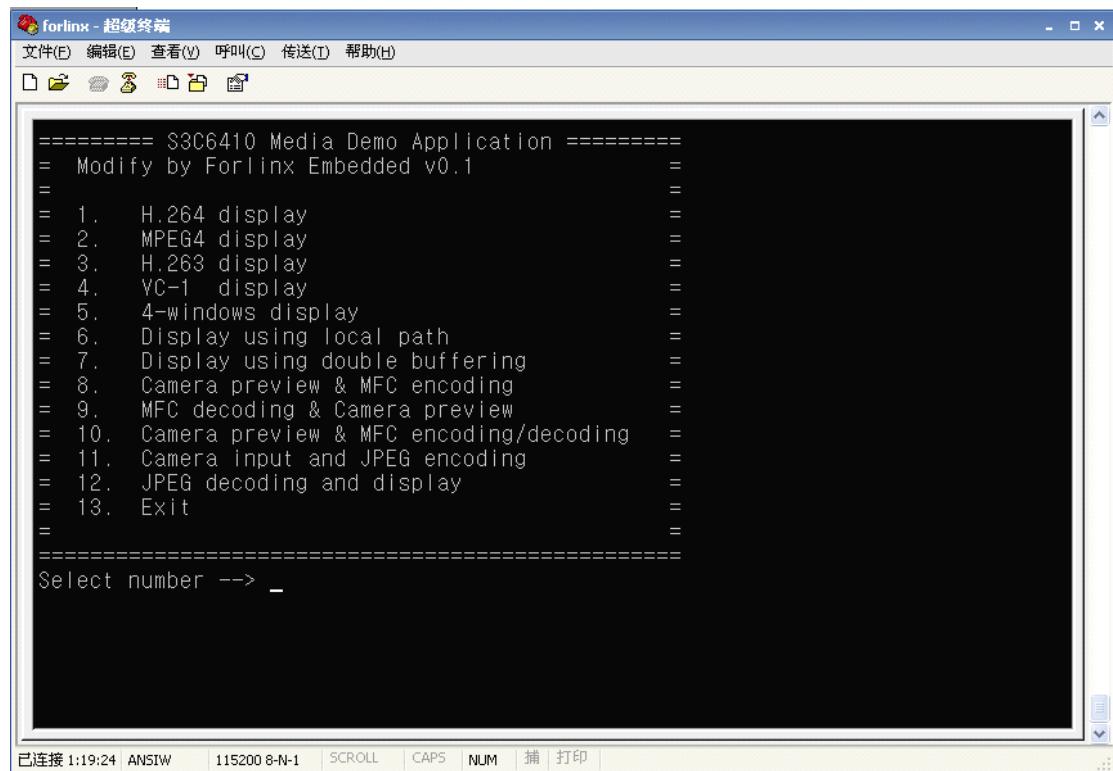
```
root@forlinx-desktop:/forlinx/Multimedia_DD# ls
APPLICATIONS CMM FIMG_2D_V1.22 JPEG_V1.01 Rotator
Changelog demo FIMV_MFC_V1.0 PP_V2.5 TVOUT_V1.1
root@forlinx-desktop:/forlinx/Multimedia_DD# cd APPLICATIONS/
root@forlinx-desktop:/forlinx/Multimedia_DD/APPLICATIONS# ls
cam_dec_preview.c capture.h display_optimization2.h jpeg display.h
cam_dec_preview.h Common display_test.c Makefile
cam_enc_dec_test.c display_4_windows.c display_test.h MFC_API
cam_enc_dec_test.h display_4_windows.h doc test.c
cam_encoder_test.c display_optimization1.c FrameExtractor TestVectors
cam_encoder_test.h display_optimization1.h JPEG_API
capture.c display_optimization2.c jpeg display.c
root@forlinx-desktop:/forlinx/Multimedia_DD/APPLICATIONS# make
```

进入 /forlinx/Multimedia_DD/APPLICATIONS 目录，#make 即开始编译。编译完成后会生成应用程序 multimediatest。

```
root@forlinx-desktop:/forlinx/Multimedia_DD/APPLICATIONS# ls
cam_dec_preview.c cam_encoder_test.o display_optimization1.c display_test.o MFC API
cam_dec_preview.h capture.c display_optimization1.h doc multimediatest
cam_dec_preview.o capture.h display_optimization1.o FrameExtractor test.c
cam_enc_dec_test.c capture.o display_optimization2.c JPEG_API test.o
cam_enc_dec_test.h Common display_optimization2.h jpeg_display.c TestVectors
cam_enc_dec_test.o display_4_windows.c display_optimization2.o jpeg_display.h
cam_encoder_test.c display_4_windows.h display_test.c jpeg_display.o
cam_encoder_test.h display_4_windows.o display_test.h Makefile
root@forlinx-desktop:/forlinx/Multimedia_DD/APPLICATIONS#
```

另外我们的文件系统里面已经包含了 multimediatest 测试程序，你需要把自己编译的 multimediatest 或者我们文件系统/usr/bin/目录下面的 multimediatest，拷贝到 SD 卡中运行，另外您还需要多媒体测试文件夹 TestVectors，TestVectors 文件夹在我们提供的光盘资料 Linux3.0\apptest\Multimedia 目录里面。

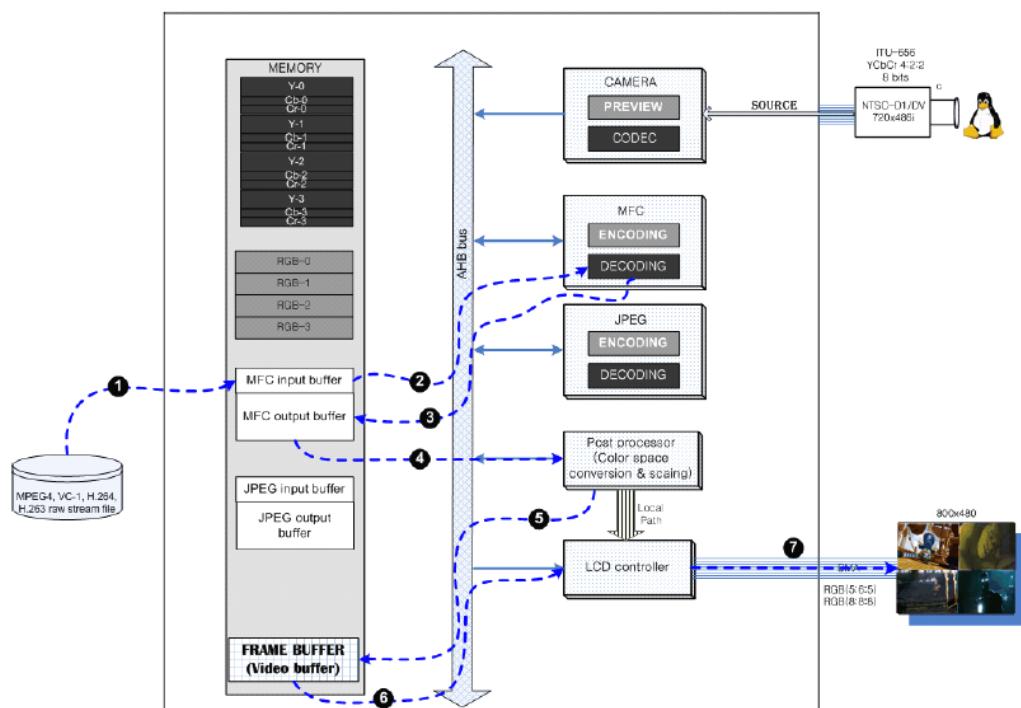
- (1) 将 multimediatest 程序和 TestVectors 文件夹拷贝到 SD 卡中
- (2) 将 SD 卡插入开发板，启动系统
- (3) 进入 SD 卡目录：#cd /sdcard
- (4) 运行测试程序：#./multimediatest 超级终端显示界面如下：



注意：第 6 项测试 Display using local path，仅支持 24 位色显示，我们的开发板目前是 RGB565 模式，是 16 位色显示，所以这里不再介绍。

9-2-2 H.264 解码

测试程序主菜单中，1-5 项均为视频解码测试，其流程如下图所示：

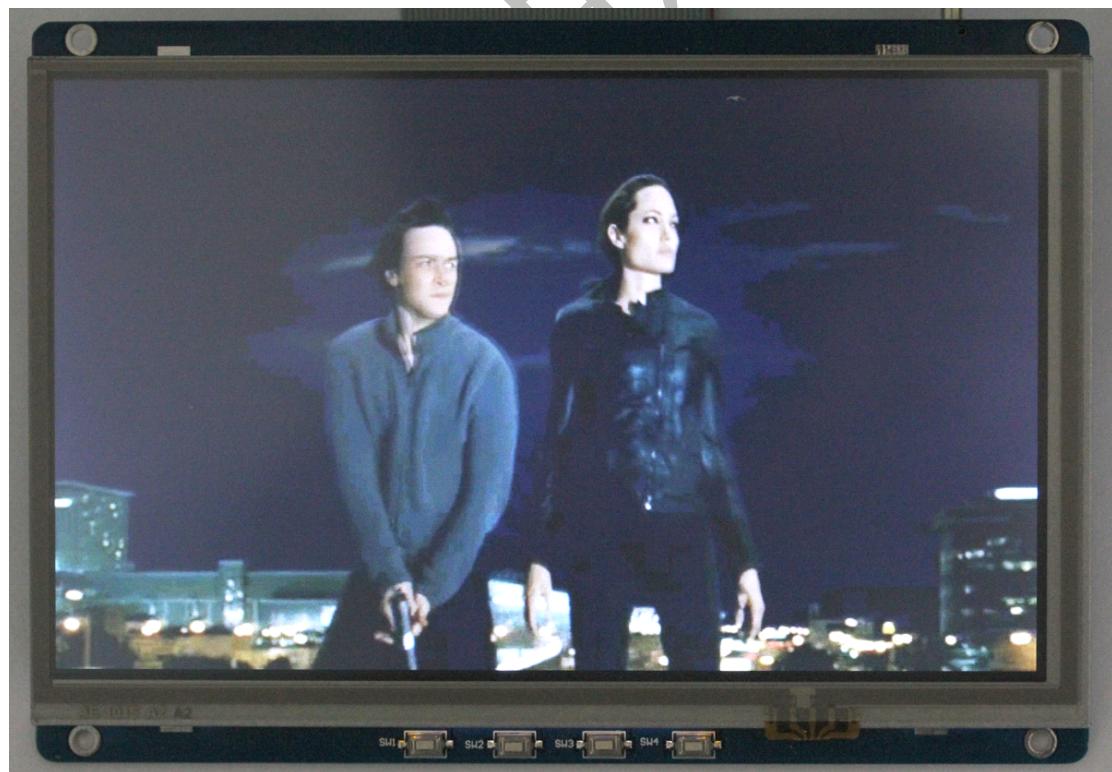


输入 ‘1’ 开始 H.264 视频解码测试：

```
Select number --> 1
=====
===== H.264 File Decodc Test =====
Forlinx Embedded, v0.1

[1. H.264 display]
Using IP       : MFC, Post processor , LCD
Input filename : wanted.264
Input vector size : VGA(640x480)
Display size   : WVGA(800x480)
Bitrate        : 971 Kbps
FPS            : 30
```

播放效果：



9-2-3 MPEG4 解码

在主菜单中输入 ‘2’，开始 MPEG4 格式视频解码测试

```
Select number --> 2
=====
MPEG4 File Decodc Test =====
Forlinx Embedded, v0.1

[2. MPEG4 display]
Using IP          : MFC, Post processor, LCD
Input filename   : shrek.m4v
Input vector size: QVGA(320x240)
Display size     : WYGA(800x480)
Bitrate         : 482 Kbps
FPS              : 24
```

播放效果：



9-2-4 H.263 解码

在主菜单中输入 ‘3’，开始 H. 263 格式视频解码测试

```
Select number --> 3
=====
===== H.263 File Decodec Test =====
Forlinx Embedded, v0.1

[3. H.263 display]
Using IP      : MFC, Post processor, LCD
Input filename : iron.263
Input vector size : QVGA(320x240)
Display size   : WVGA(800x480)
Bitrate       : 460 Kbps
FPS           : 30
```

播放效果：



9-2-5 VC-1 解码

在主菜单中输入 ‘4’，开始 VC-1 格式视频解码测试

```
Select number --> 4
=====
VC-1 File Decodec Test -----
Forlinx Embedded, v0.1

[4. VC-1 display]
Using IP      : MFC, Post processor, LCD
Input filename : test2_0.rcv
Input vector size : QVGA(320x240)
Display size   : WVGA(800x480)
Bitrate       : 460 Kbps
FPS           : 30
```

在主菜单中输入 ‘4’，开始VC-1 格式视频解码测试
播放效果：



9-2-6 多种视频同时解码

在主菜单中输入 ‘5’，开始多种格式视频同时解码测试

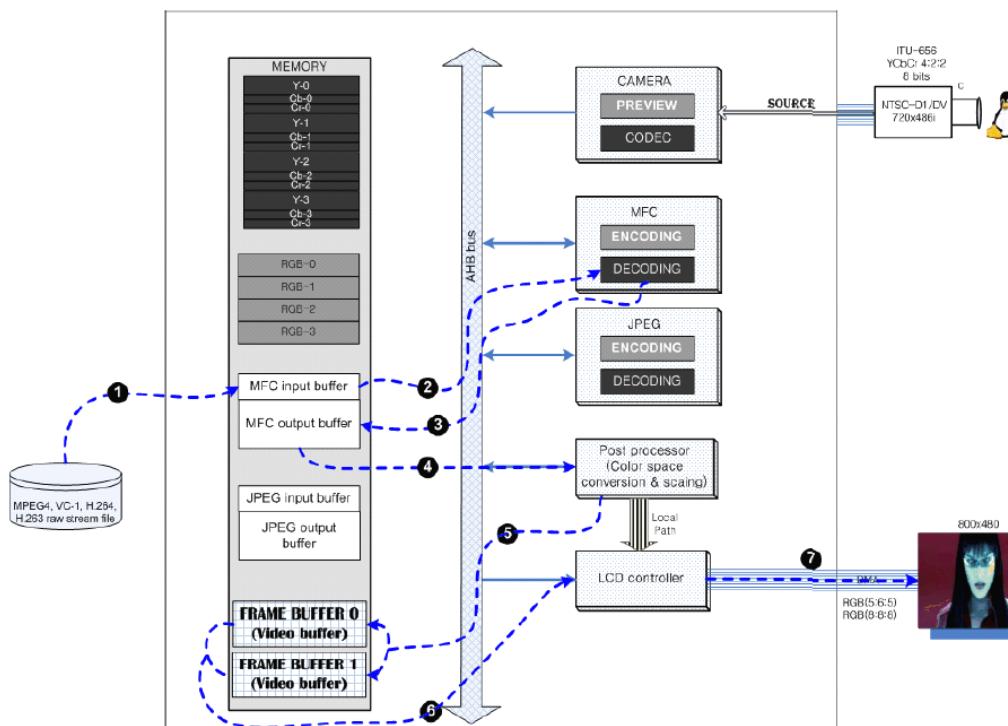
```
[4-windows display] Forlinx Embedded, v0.1
Using IP : MFC, Post processor, LCD
*****
* Frame buffer      : 0          * Frame buffer      : 1          *
* Codec            : H.264       * Codec            : MPEG4       *
* Input filename   : veggie.264 * Input filename   : shrek.m4v   *
* Input vector size: QVGA       * Input vector size: QVGA       *
* Display size     : 400x240    * Display size     : 400x240    *
* Bitrate          : 460 Kbps    * Bitrate          : 482 Kbps    *
* FPS              : 30          * FPS              : 24          *
*
*****
* Frame buffer      : 2          * Frame buffer      : 3          *
* Codec            : H.263       * Codec            : VC-1        *
* Input filename   : iron.263  * Input filename   : test2_0.rcv *
* Input vector size: QVGA       * Input vector size: QVGA       *
* Display size     : 400x240    * Display size     : 400x240    *
* Bitrate          : 460 Kbps    * Bitrate          : 460 Kbps    *
* FPS              : 30          * FPS              : 30          *
*
*****
```

播放效果：



9-2-7 H.264 解码&LCD 双缓冲

该程序主要测试双视频缓冲，流程如下图所示：



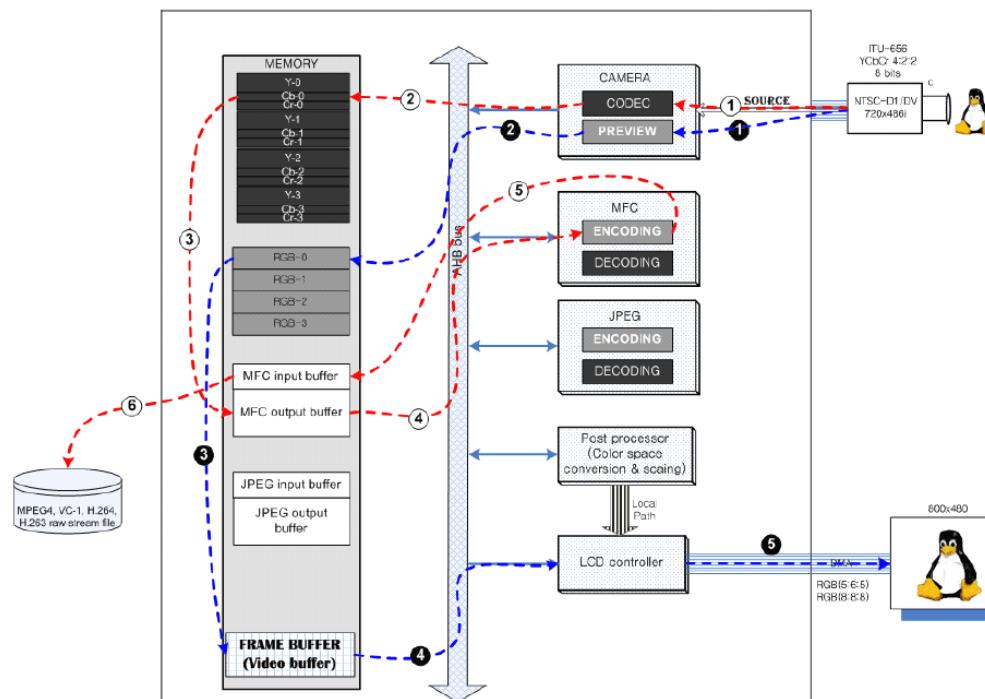
在主菜单中输入 7，开始本项测试，效果与 4-1 一致。

```
Select number --> 7

[7. Display using double buffering]
Forlinx Embedded, v0.1
Using IP : WVGA(800x480)
Bitrate : 971 Kbps
FPS : 30
```

9-2-8 摄像头预览&MFC 编码

该程序执行摄像头的预览，并同时对当前帧进行压缩；下图中蓝色线条代表预览流程，红色为将当前 codec 通道数据进行编码，两部分在不同的线程中独立执行。



在主菜单中输入 ‘8’，开始本项测试：

```
Select number --> 8

[8. Camera preview & MFC encoding]
Forlinx Embedded, v0.1
Using IP          : MFC, Post processor, LCD, Camera
Display size     : VGA(640x480)

e : Encoding
x : Exit
Select ==>
```

输入 ‘e’ 进行编码，长度为100 帧。

输入 ‘x’ 退出。

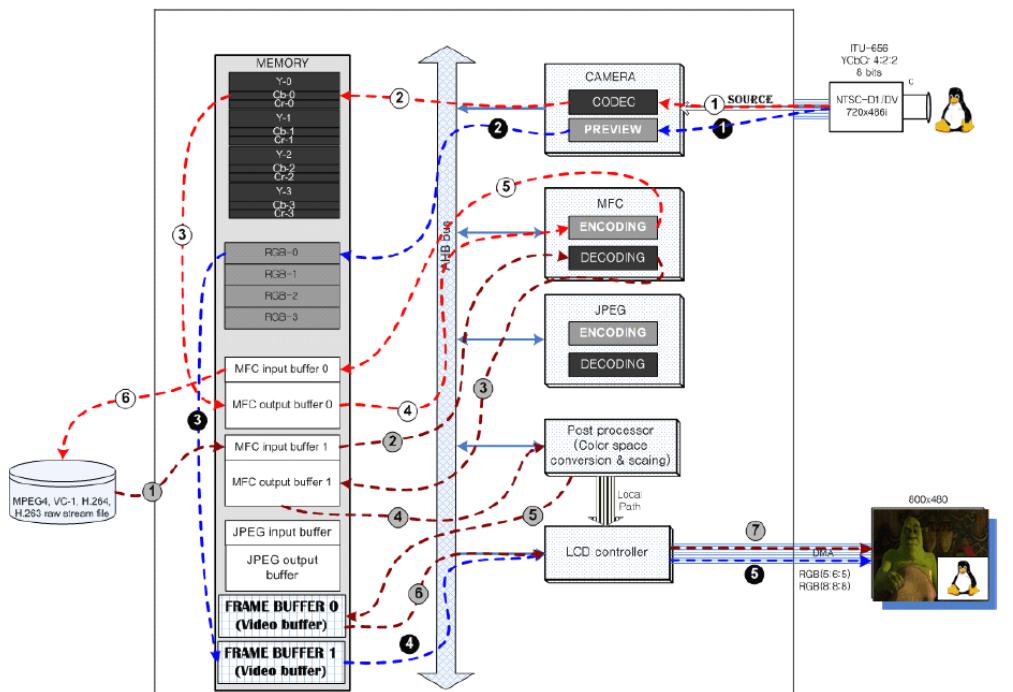
摄像头采集效果：



禁行

9-2-9 H.264 解码&摄像头预览

该程序同时执行 H.264 的解码和摄像头预览，并且可以对当前帧进行编码



在主菜单中输入 ‘9’，开始本项测试：

```
Select number --> 9

[9. MFC decoding & Camera preview]
Forlinx Embedded, v0.1
Using IP : MFC, Post processor, LCD, Camera
Camera preview size : QVGA(320x240)
Display size : WVGA(800x480)

e : Encoding
x : Exit
Select ==>
```

输入 ‘e’ 进行编码，长度为100 帧。

输入 ‘x’ 退出

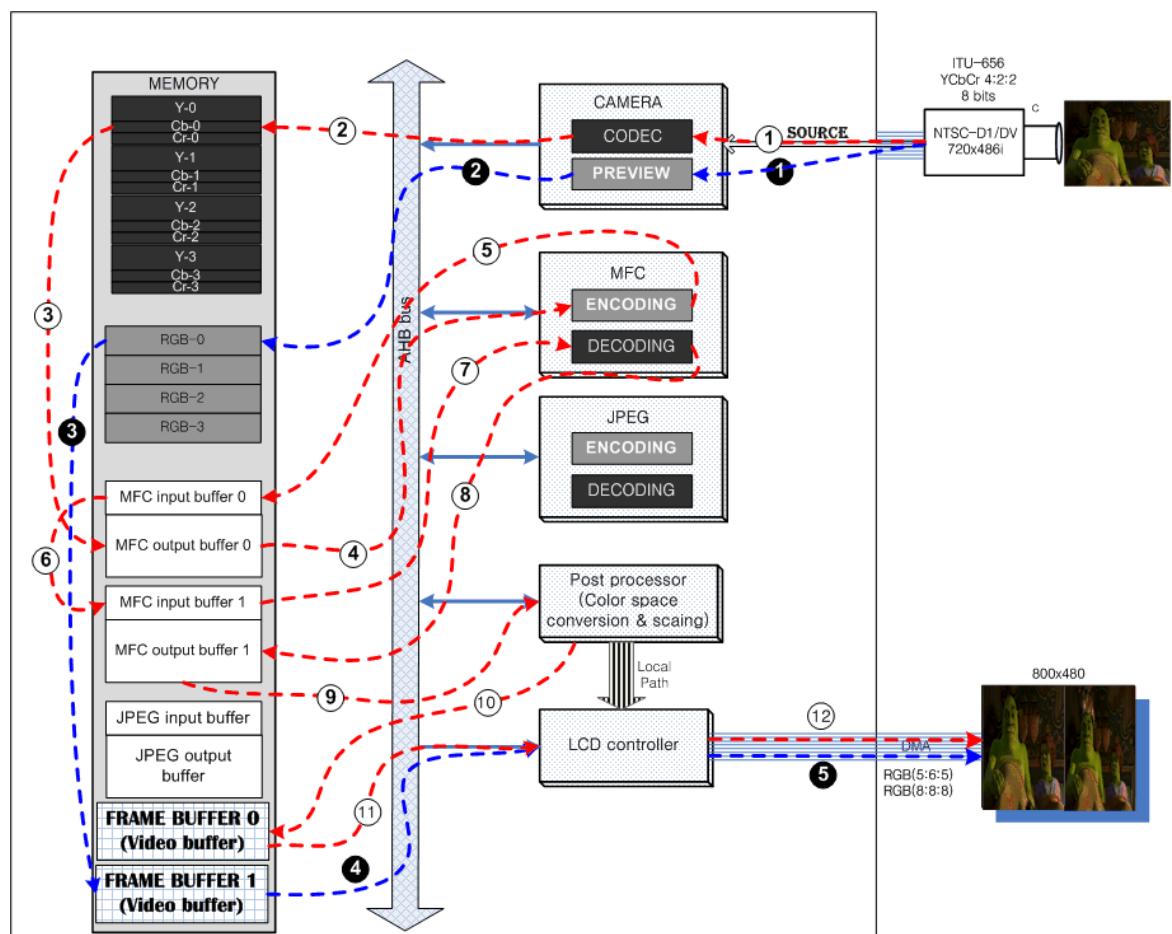
效果图：



上 一
游 网

9-2-10 摄像头预览&MFC 编码解码

该程序同时执行摄像头预览、编码、解码



在主菜单中输入‘10’，开始本项测试：
注：若LCD右侧显示异常，请重启测试程序

```
Select number --> 10
[10. Camera preview & MFC encoding/decoding]
Forlinx Embedded, v0.1
Using IP           : MFC, Post processor, LCD, Camera
Camera preview size : (400x480)
Display size       : (400x480)

x : Exit
Select ==> _
```

输入‘x’退出

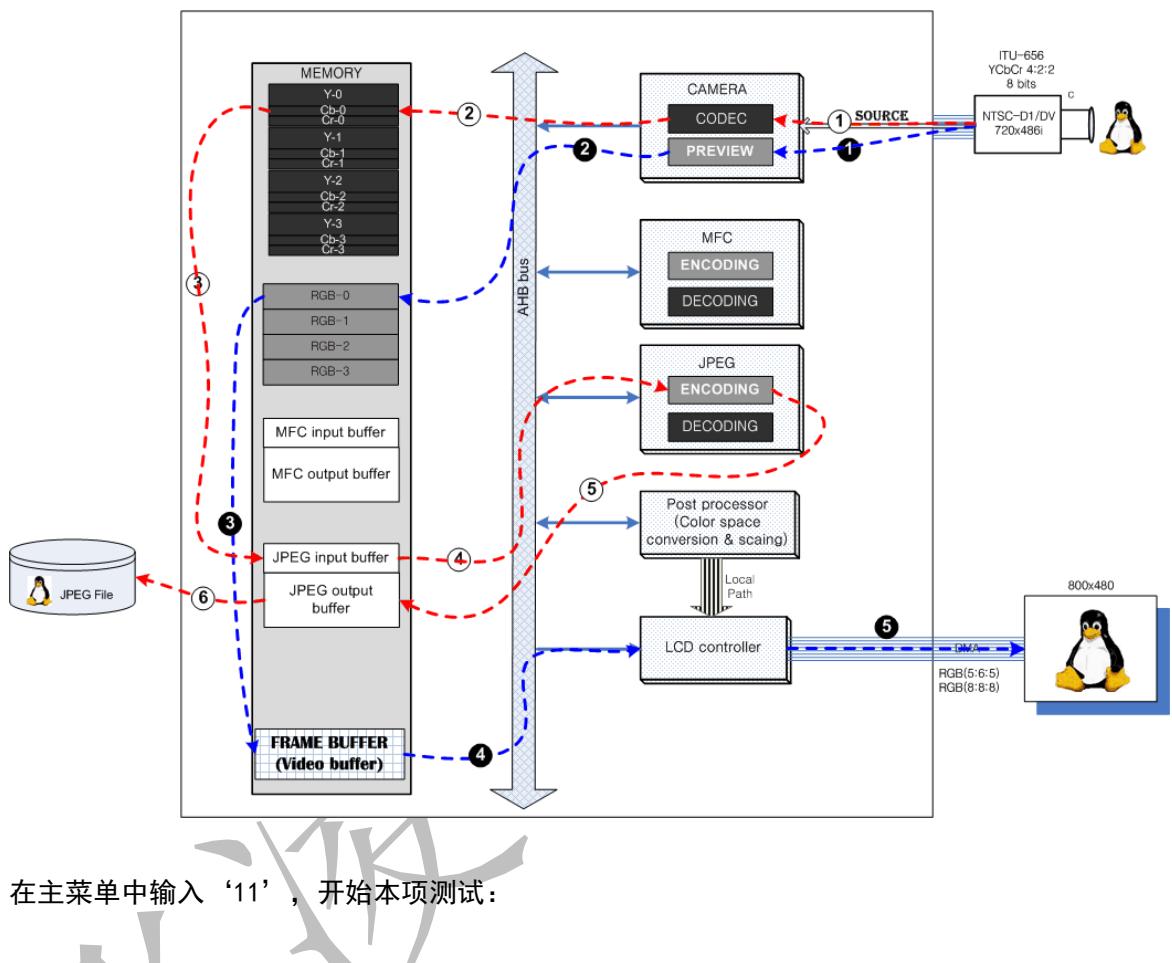
效果图如下，其中右侧为预览效果，左侧为编码后再解码效果



水印

9-2-11 摄像头预览&JPEG 编码

该程序执行摄像头预览，并可对当前帧进行 JPEG 编码



```
Select number --> 11
[11. Camera input & JPEG encoding]
Forlinx Embedded, v0.1
Using IP           : Post processor, LCD, Camera, JPEG
Camera preview size : VGA(640x480)
Capture size       : VGA(640x480)

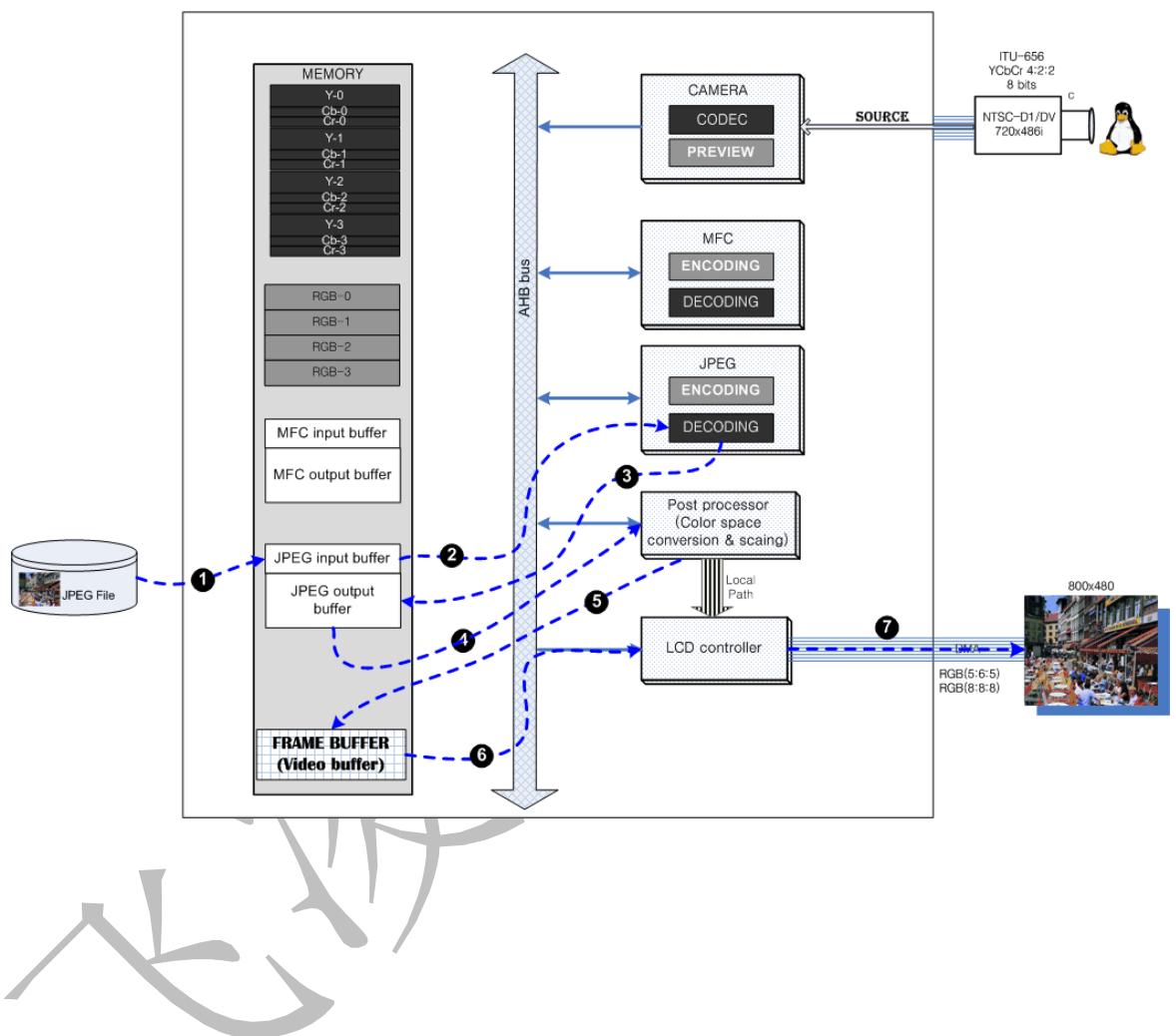
c : Capture
x : Exit
Select ==>
```

输入 ‘c’ 抓取图片

输入 ‘x’ 退出

9-2-12 JPEG 解码

该程序对 JPEG 文件进行解码，并在 LCD 上显示



在主菜单中输入 ‘12’ , 开始本项测试, 效果如下图所示:



水
游
世

9-3 多媒体综合测试二

以上测试基于三星提供的源代码，只能播放固有的视频文件，普通的 AVI 和 MP4 文件是不能播放的，我们制作了一个开源播放器，该播放器使用 6410 的硬解码功能解码播放符合 6410 格式的媒体文件，播放器有命令行和 Qt 图形界面两个版本，且完全开源。

使用播放器播放测试视频时，需要把测试视频拷贝到 SD 卡里面，然后把 SD 卡插入到我们的开发板中。我们的测试视频位于多媒体测试文件夹内，里面有 H.264 编码格式的 AVI 文件，和 MPEG4 编码格式的 MP4 文件。

注意：我们的 AVI 和 MP4 测试视频是 6410 CPU 支持的格式，如果是您自己的多媒体文件需要使用转换工具转换成 6410 所支持的，转换软件我们会在下面的章节说明。

下面我们先看一下播放效果。

9-3-1 命令行的开源播放软件 **player**

player 软件是 6410 平台上的一个开源软件，源代码为 s3c6410-multiplayer.tar.gz，位于我们的光盘资料中的多媒体测试目录下，该软件使用 6410 的硬件解码特性，解码播放 AVI 和 MP4 文件，您可以从 Google Code 上面下载最新版本，下载链接：

<http://code.google.com/p/s3c6410-multiplayer/>

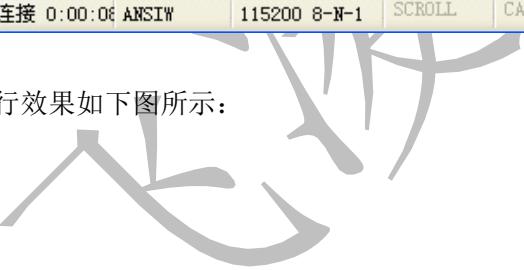
我们的文件系统里面已经有了该播放器的可执行文件 player，位于 /bin 目录下面，该软件只适用于 4.3 寸屏显示。

如图所示：



```
TT - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
[root@FORLINX6410]#
[root@FORLINX6410]# ls /bin/player
/bin/player
[root@FORLINX6410]# _
```

播放 AVI 文件,《黑客帝国》片段:



```
TT - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
[root@FORLINX6410]#
[root@FORLINX6410]# player /sdcard/TestVideo/H.264/TH\ MATRIX.avi
```

运行效果如下图所示:



播放 MP4 文件，《泰坦尼克号》片段：

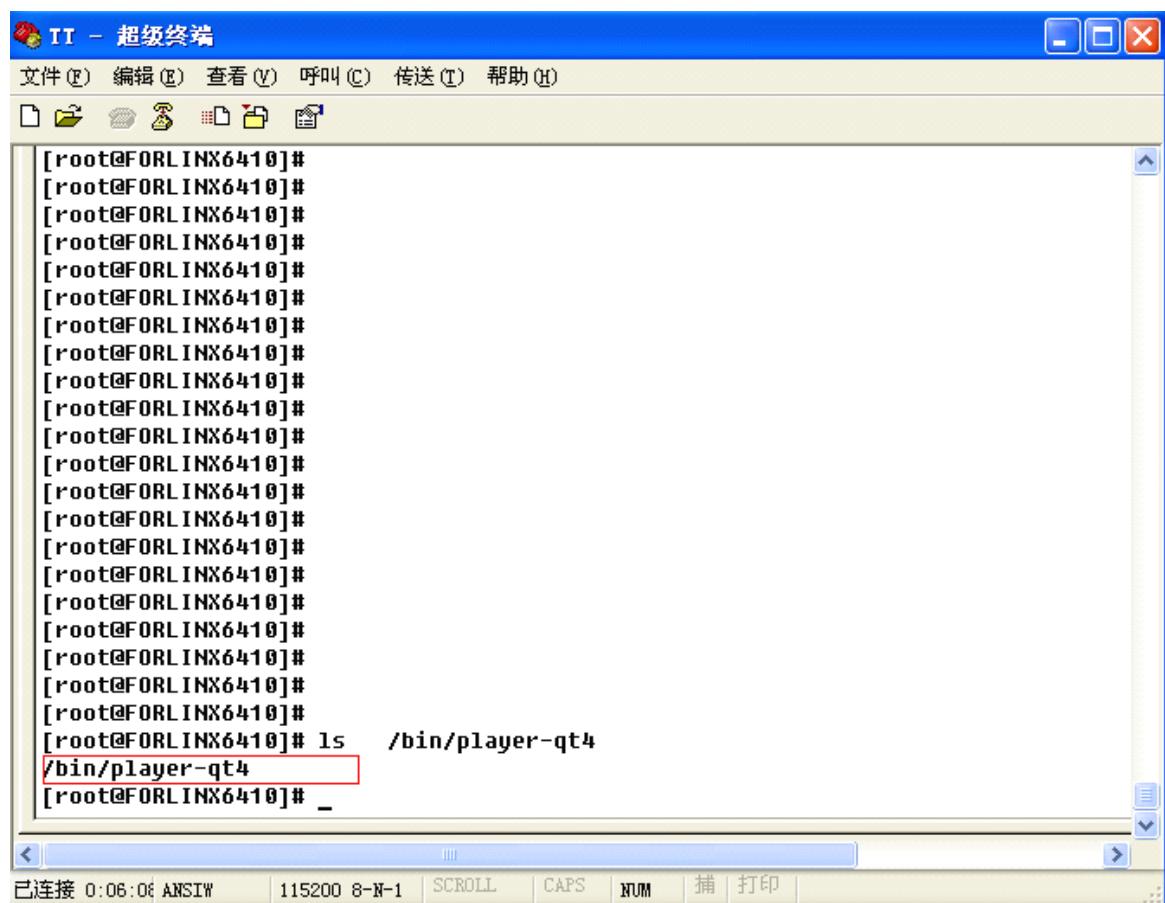
运行效果如下图所示：



播放过程中可以点击屏的不同位置，控制视频的播放，如暂停，退出。

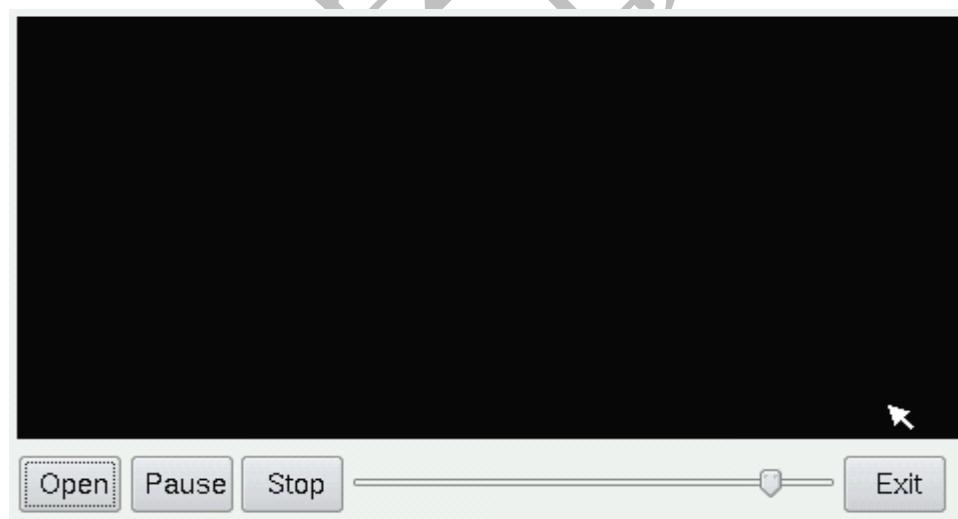
9-3-2 Qt 图形界面程序 **player-qt4**

player-qt4 是我们在开源播放器软件 player 的基础上加上 Qt4.7 界面库的一个播放器，我们的光盘资料里面提供了这个播放器的源代码，您可以随意修改和使用。我们发布的文件系统里面已经有了这个播放器的二进制文件，位于 /bin 目录，另外 player-qt4 只适用于 4.3 寸屏显示，如下图所示：

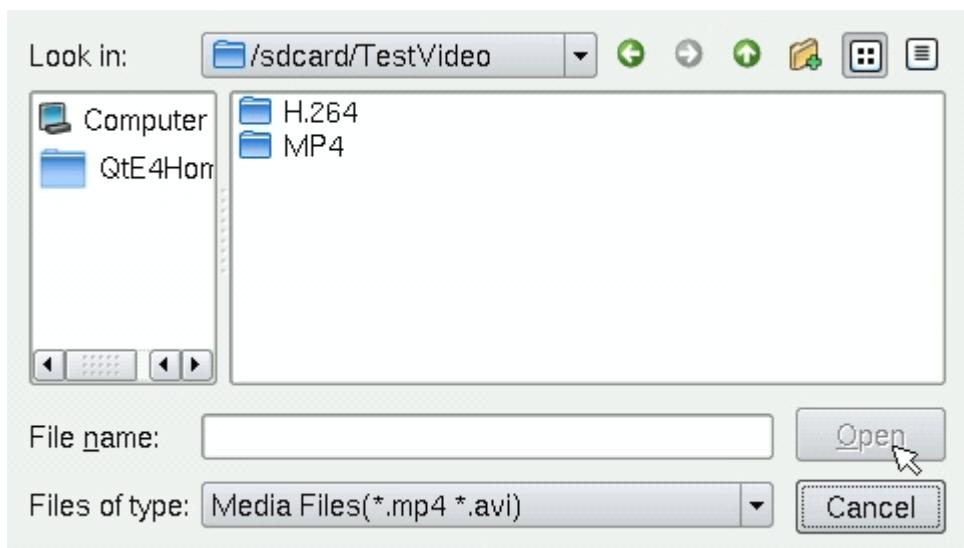


```
[root@FORLINUX6410]#  
[root@FORLINUX6410]# ls /bin/player-qt4  
/bin/player-qt4  
[root@FORLINUX6410]# _
```

运行 player-qt4，屏幕上会出现播放界面，如图所示：



点击打开按钮，选择您播放的视频文件。



您可以单击界面上的按钮来控制视频的播放，比如暂停，快进，退出。

9-3-3 如何转换普通的多媒体文件到 6410 支持播放格式

Aimersoft iPhone Converter Suite 软件可以将多媒体视频转换成 6410 上可硬解播放的 MP4 视频格式。MediaCoder 软件可以将多媒体视频转换成 6410 上可硬解播放的 H264 视频格式，我们提供的测试视频就是用这两个软件转换的。

用户可以从网下载最新版本，这里不再详述。

第十章 Linux 基础实验教程

10-1 实验一 shell 编程

1、实验目的:

- 了解什么是 shell
- 掌握 shell 编程

2、实验内容:

编程实现将输入的字符串输出

3、实验设备:

PC 机 (Linux 环境)、飞凌 6410 开发板

4、预备知识:

● Shell 简介

操作系统与外部最主要接口就叫做 shell。shell 是操作系统最外的一层。shell 管理你与操作系统之间的交互：等待你输入，向操作系统解释你的输入，并且处理各种各样的操作系统的输出结果。

用户登录或运行终端类比程序时，实际进入了 Shell。那么，Shell 是什么呢？确切一点说，Shell 就是一个命令行解释器，它的作用就是遵循一定的语法规则将输入的命令加以解释并传给系统。它为用户提供了一个向 Linux 发送请求以便运行程序的接口系统级程序，用户可以用 Shell 来启动、挂起、停止甚至是编写一些程序。

Shell 本身是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。作为命令语言，它互动式地解释和执行用户输入的命令；作为

程序设计语言，它定义了各种变量和参数，并提供了许多在高阶语言中才具有的控制结构，包括循环和分支。它虽然不是 Linux 系统内核的一部分，但它调用了系统内核的大部分功能来执行程序、创建文档并以并行的方式协调各个程序的运行。因此，对于用户来说，Shell 是最重要的实用程序，深入了解和熟练掌握 Shell 的特性及其使用方法，是用好 Linux 系统的关键。可以说，Shell 使用的熟练程度反映了用户对 Linux 使用的熟练程度。

当用户使用 Linux 时是通过命令来完成所需工作的。一个命令就是用户和 Shell 之间对话的一个基本单位，它是由多个字符组成并以换行结束的字串。

● 交互与非交互式 shell

交互式模式就是 shell 等待你的输入，并且执行你提交的命令。这种模式被称作交互式是因为 shell 与用户进行交互。这种模式也是大多数用户非常熟悉的：登录、执行一些命令、签退。当你签退后，shell 也终止了。

shell 也可以运行在另外一种模式：非交互式模式。在这种模式下，shell 不与你进行交互，而是读取存放在文件中的命令，并且执行它们。当它读到文件的结尾，shell 也就终止了。

● Shell 类型

在 UNIX 中主要有两大类 shell

Bourne shell (包括 sh, ksh, and bash)

Bourne shell (sh)

Korn shell (ksh)

Bourne Again shell (bash)

POSIX shell (sh)

C shell (包括 csh and tcsh)

C shell (csh)

TENEX/TOPS C shell (tcsh)

Bourne Shell

最初的 **UNIX shell** 是由 Stephen R. Bourne 于 20 世纪 70 年代中期在新泽西的 AT&T 贝尔实验室编写的，这就是 **Bourne shell**。**Bourne shell** 是一个交换式的命令解释器和命令编程语言。**Bourne shell** 可以运行为 **login shell** 或者 **login shell** 的子 **shell**(**subshell**)。只有 **login** 命令可以调用 **Bourne shell** 作为一个 **login shell**。此时，**shell** 先读取/etc/profile 文件和 \$HOME/.profile 文件。/etc/profile 文件为所有的用户定制环境，\$HOME/.profile 文件为本用户定制环境。最后，**shell** 会等待读取你的输入。

C Shell

Bill Joy 于 20 世纪 80 年代早期，在 Berkeley 的加利福尼亚大学开发了 **C shell**。它主要是为了让用户更容易的使用交互式功能，并把 **ALGOL** 风格的语法结构变成了 **C** 语言风格。它新增了命令历史、别名、文件名替换、作业控制等功能。

Korn Shell

有很长一段时间，只有两类 **shell** 供人们选择，**Bourne shell** 用来编程，**C shell** 用来交互。为了改变这种状况，AT&T 的 bell 实验室 David Korn 开发了 **Korn shell**。**ksh** 结合了所有的 **C shell** 的交互式特性，并融入了 **Bourne shell** 的语法。因此，**Korn shell** 广受用户的欢迎。它还新增了数学计算、进程协作(**coprocess**)、行内编辑(**inline editing**)等功能。

Korn Shell 是一个交互式的命令解释器和命令编程语言，它符合 **POSIX**——一个操作系统的国际标准。**POSIX** 不是一个操作系统，而是一个目标在于应用程序的移植性的标准——在源程序一级跨越多种平台。

Bourne Again Shell (bash)

bash 是 **GNU** 计划的一部分，用来替代 **Bourne shell**。它用于基于 **GNU** 的系统如 **Linux**。大多数的 **Linux**(**Red Hat**, **Slackware**, **Caldera**)都以 **bash** 作为缺省的 **shell**，并且运行 **sh** 时，其实调用的是 **bash**。

POSIX Shell

POSIX shell 是 **Korn shell** 的一个变种。当前提供 **POSIX shell** 的最大卖主是

Hewlett-Packard。在 HP-UX 11.0, POSIX shell 就是 /bin/sh, 而 bsh 是 /usr/old/bin/sh
各主要操作系统下缺省的 shell:

AIX 下是 Korn Shell.

Solaris 和 FreeBSD 缺省的是 Bourne shell.

HP-UX 缺省的是 POSIX shell.

Linux 是 Bourne Again shell.

● 什么是脚本

脚本是批处理文件的延伸，是一种纯文本保存的程序，一般说来的计算机脚本程序是确定的一系列控制计算机进行运算操作动作的组合，在其中可以实现一定的逻辑分支等。

脚本程序相对一般程序开发来说比较接近自然语言，可以不经编译而是解释执行，利于快速开发或一些轻量的控制。

本质上，shell script 是命令行命令简单的组合到一个文件里面。

● Shell 特点

用户与 Linux 的接口

命令解释器

支持多用户

支持复杂的编程语言

Shell 有很多种，如：csh, tcsh, pdksh, ash, sash, zsh, bash 等

Linux 的缺省 Shell 为 bash(Bourne Again Shell)

● 创建脚本文件遵循的步骤

1、使用编辑器加载文件

2、确认脚本文件的第一行是：#! /bin/bash

3、保存脚本文件并，退出编辑器

4、使用 “chmod u+x 脚本文件名”，标注脚本

文件的可执行属性

5、使用 “./脚本文件”，执行脚本

5、实验代码

```
#!/bin/sh  
echo -n "please input your name:"  
read name  
echo "thanks,"  
echo $name
```

6、运行

Shell 文件不用编译可以直接运行

A、PC 机上运行

```
[root@mdl-desktop:/usr/test/shell# ./dircmp  
please input your name:forlinx  
thanks,  
Hello forlinx  
root@mdl-desktop:/usr/test/shell# ]
```

B、开发板上运行

```
[root@TE6410 /tmp]# chmod +x dircmp  
[root@TE6410 /tmp]# ./dircmp  
please input your name: forlinx  
thanks,  
Hello forlinx  
[root@TE6410 /tmp]# ]
```

10-2 实验二 Hello world

1实验目的:

了解交叉编译和本机编译的区别

2实验内容

串口输出“Hello world”

3实验设备:

PC 机 (Linux 环境)、飞凌 6410 开发板

4实验代码

```
*****hello.c*****  
*****/  
#include <stdio.h>  
Main()
```

```
{  
    Printf("forlinx--Hello world!");  
}
```

5 编译及运行

A 本机编译。

本机编译简单说就是在什么环境下编译就在什么环境下运行。我们这个在 PC 机 Linux 环境下运行，当然也是运行在 PC 机 Linux 环境下。编译命令如下：

```
gcc -o hello-pc hello.c //该命令得到的可执行文件为 hello
```

运行可执行程序

```
./hello-pc
```

输出 forlinx—Hello world!

```
root@mdl-desktop:/usr/test/hello# gcc -o hello-pc hello.c  
root@mdl-desktop:/usr/test/hello# ./hello-pc  
forlinx--Hello world!  
root@mdl-desktop:/usr/test/hello#
```

B 交叉编译。

交叉编译实际上就是从 A 环境下编译，在 B 环境下运行。我们是在 PC 机 Linux 环境下编译在飞凌 6410 开发板上运行。编译命令如下：

```
/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc -o hello-arm hello.c
```

编译完成后，得到 hello-arm 可执行文件。这个文件不能再 PC 上运行

```
root@mdl-desktop:/usr/test/hello# /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc -o hello-arm hello.c  
root@mdl-desktop:/usr/test/hello# ./hello-arm  
bash: ./hello-arm: cannot execute binary file  
root@mdl-desktop:/usr/test/hello#
```

如果此时在 PC 上执行改成虚怎会提示如上图：“不可执行的二进制文件”。

现将 hello-arm 可执行文件通过 SD 卡（或 U 盘）拷贝到开发板中，在飞凌开发板上执行过程如下图：

```
[root@TE6410 /]# cd /tmp  
[root@TE6410 /tmp]# ls  
[root@TE6410 /tmp]# [root@TE6410 /tmp]#  
[root@TE6410 /tmp]# ls  
hello-arm  
[root@TE6410 /tmp]# ./hello-arm  
- /bin/sh: ./hello-arm: Permission denied  
[root@TE6410 /tmp]# chmod +x hello-arm  
[root@TE6410 /tmp]# ./hello-arm  
forlinx--Hello world!  
[root@TE6410 /tmp]#
```

10-3 实验三 多线程实验

1、实验目的:

- 了解线程
- 学会多线程编程

2、实验内容:

多线程编程，实现线程间循环运行

3、实验设备:

PC 机（Linux）、飞凌 6410 开发板

4、预备知识:

Linux 是一个**多用户、多任务**的操作系统。多用户是指多个用户可以在同一时间使用计算机系统；**多任务**是指 Linux 可以同时执行几个任务，它可以在还未执行完一个任务时又执行另一项任务。在操作系统设计上，从进程（Process）演化出线程（Thread），最主要的目的就是更好地支持多处理器，并且减小（进程/线程）上下文切换的开销。

进程和线程的关系

根据操作系统的定义，**进程是系统资源管理的最小单位**，**线程是程序执行的最小单位**。线程和进程十分相似，不同的只是线程比进程小。

首先，线程采用了多个线程可共享资源的设计思想。例如，它们的操作大部分都是在同一地址空间进行的。其次，从一个线程切换到另一线程所花费的代价比进程低。再次，进程本身

的信息在内存中占用的空间比线程大。

因此，线程更能充分地利用内存。线程可以看作是在进程内部执行的指定序列。线程和进程的最大区别在于线程完全共享相同的地址空间，运行在同一地址上。

Linux 线程的定义

线程是在共享内存空间中并发的多道执行路径，它们共享一个进程的资源，如文件描述和信号处理。在两个普通进程(非线程)间进行切换时，内核准备从一个进程的上下文切换到另一个进程的上下文要花费很大的开销。

这里上下文切换的主要任务是保存老进程 CPU 状态，并加载新进程的保存状态，用新进程的内存映像替换老进程的内存映像。线程允许进程在几个正在运行的任务之间进行切换，而不必执行前面提到的完整的上下文。

Linux 进程和线程技术发展

线程技术早在 20 世纪 60 年代就被提出，但真正应用多线程到操作系统中还是在 20 世纪 80 年代中期。现在，多线程技术已经被许多操作系统所支持，包括 Windows NT/2000 和 Linux。

在 1999 年 1 月发布的 Linux 2.2 内核中，进程是通过系统调用 `fork` 创建的，新的进程是原来进程的子进程。需要说明的是，在 Linux 2.2.x 中，不存在真正意义上的线程，Linux 中常用的线程 `Pthread` 实际上是通过进程来模拟的。

也就是说，Linux 中的线程也是通过 `fork` 创建的，是“轻”进程。Linux 2.2 缺省只允许 4096 个进程/线程同时运行，而高端系统同时要服务上千的用户，所以这显然是一个问题。它一度是阻碍 Linux 进入企业级市场的一大因素。

2001 年 1 月发布的 Linux 2.4 内核消除了这个限制，并且允许在系统运行中动态调整进程数上限。因此，进程数现在只受制于物理内存的多少。在高端服务器上，即使只安装了 512MB 内存，现在也能轻而易举地同时支持 1.6 万个进程。

在 Linux 2.5 内核中，已经做了很多改进线程性能的工作。在 Linux 2.6 中改进的线程模型仍然是由 Ingo Molnar 来完成的。它基于一个 1:1 的线程模型（一个内核线程对应一个用户线程），包括内核内在的对新 NPTL (Native Posix Threading Library) 的支持，这个新的 NPTL 是由 Molnar 和 Ulrich Drepper 合作开发的。

2003 年 12 月发布的 Linux 2.6 内核，对进程调度经过重新编写，去掉了以前版本中效率

不高的算法。进程标识号（**PID**）的数目也从 3.2 万升到 10 亿。内核内部的大改变之一就是 **Linux** 的线程框架被重写，以使 **NPTL** 可以运行其上。

Linux 的另一种可选线程模型是 IBM 开发的 **NGPT**（*Next Generation Posix Threads for Linux*），它是基于 **GNU Pth**（*GNU Portable Threads*）项目而实现的 **M:N** 模型（**M** 个用户态线程对应 **N** 个核心态线程）。

NPTL 的设计目标可归纳为以下几点：**POSIX** 兼容性、**SMP** 结构的利用、低启动开销、低链接开销（即不使用线程的程序不应当受线程库的影响）、与 **LinuxThreads** 应用的二进制兼容性、软硬件的可扩展能力、多体系结构支持、**NUMA** 支持，以及与 **C++** 集成等。

对于运行负荷繁重的线程应用 **Pentium Pro** 及更先进的处理器而言，这些是主要的性能提升，也是企业级应用中很多高端系统一直以来所期待的。线程框架的改变包含 **Linux** 线程空间中的许多新的概念，包括线程组、线程各自的本地存储区、**POSIX** 风格信号，以及其他改变。改进后的多线程和内存管理技术有助于更好地运行大型多媒体应用软件。

Linux 线程实现方法

目前线程有用户态线程和核心态线程两种方法实现。

1. 用户态线程

用户态线程是一个精细的软件工具，允许多线程的程序运行时不需要特定的内核支持。如果一个进程中的某一个线程调用了一个阻塞的系统调用，则该进程就会被阻塞，该进程中的其它所有线程也同时被阻塞。因此，**Unix** 使用了异步 I/O 机制。这种机制主要的缺点在于，在一个进程中的多个线程调度中无法发挥多处理器的优势（如上述的阻塞情况）。

用户态线程优点如下：

- ◆某些线程操作的系统消耗大大减少。比如，对属于同一个进程的线程之间进行调度切换时，不需要调用系统调用，因此将减少额外的消耗，一个进程往往可以启动上千个线程。
- ◆用户态线程的实现方式可以被定制或修改，以适应特殊应用的要求。它对于多媒体实时过程等尤其有用。另外，用户态线程可以比核心态线程实现方法默认情况支持更多的线程。

2. 核心态线程

核心态线程的实现方法允许不同进程中的线程按照同一相对优先调度方法进行调度，这样有利于发挥多处理器的并发优势。

目前，线程主要的实现方法是用户态线程。有几个研究项目已经实现了一些核心态线程的形式，其中比较著名的是 MACH 分布式操作系统。

通过允许用户代码对内核线程调度的参与，该系统将用户态和核心态两种线程实现方法的优点结合了起来。通过提供这样一个两级调度机制，内核在保留了对处理器时间分配控制的同时，也使一个进程可以充分利用多处理器的优势。

Linux 对超线程技术支持

超线程技术（Hyperthreading Technology）是 Intel 公司的创新设计。HT 技术就是利用特殊的硬件指令，把两个逻辑内核模拟成两个物理芯片，让单个处理器都能使用线程级并行计算，从而兼容多线程操作系统和软件，并提高处理器的性能。

操作系统或应用软件的多线程可以同时运行于一个处理器上，两个逻辑处理器共享一组处理器执行单元，并行完成加、乘、负载等操作。在同一时间里，应用程序可以使用芯片的不同部分。虽然单线程芯片每秒钟能够处理成千上万条指令，但是在任一时刻只能够对一条指令进行操作。而 HT 技术可以使芯片同时进行多线程处理，当在支持多处理器的 Windows XP 或 Linux 等操作系统之下运行时，同时运行多个不同的软件程序可以获得更高的运行效率。这两种方式都可使计算机用户获得更优异的性能和更短的等待时间。

Linux 是第一个把超线程特性引入市场的操作系统，它在发布 2.4.17 内核时，就开始包含对 Intel P4 处理器的超线程的支持（Linux 2.4 内核最初的发布版本中不支持），它包括以下增强技术：

- ◆ 128 字节锁对齐。
- ◆ 螺旋等待循环优化。
- ◆ 基于非执行的延迟循环。
- ◆ 检测支持超线程的处理器，并启动逻辑处理器，如同该机器是 SMP（多处理器构架）。
- ◆ MTRR 和微码更新（Microcode Update）驱动程序中的串行化，因为它们影响共享状态。
- ◆ 在逻辑处理器调度发生之前，当系统空闲时对物理处理器上的调度进行优先级排序时，对调度程序进行优化。

◆偏移用户堆栈以避免 64K 混叠。

以上简单介绍了 Linux 线程的属性、与进程关系、线程的实现方法，以及 Linux 对超线程技术支持等，希望对大家了解 Linux 线程技术有所帮助。

5、实验代码

```
#include<stddef.h>
#include<unistd.h>
#include<pthread.h>
thread1()
{
    while(1)
    {
        printf("I am the thread\n");
        sleep(1);
    }
}
main()
{
    int ret;
    pthread_t id;
    ret(pthread_create(&id, NULL, (void*)thread1, NULL));
    if(ret!=0)
    {
        printf("thread_send error\n");
        return(1);
    }
    while(1)
    {
        printf("I am the main-thread\n");
        sleep(1);
    }
}
```

5 编译及运行

A 本机编译

```
gcc -o thread-pc thread.c -lpthread
运行./thread-pc
```

```
root@mdl-desktop:/usr/test/thread# ls  
thread.c  
root@mdl-desktop:/usr/test/thread# gcc -o thread-pc thread.c -lpthread  
root@mdl-desktop:/usr/test/thread# ./thread-pc  
I am the thread  
I am the main-thread  
^C  
root@mdl-desktop:/usr/test/thread#
```

B 交叉编译

/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc -o thread-arm thread.c -lpthread
将得到的可执行程序传送到开发板上运行。结果与在 PC 机上运行基本是一样的。

```
[root@TE6410 /tmp]# ls  
hello-arm  thread-arm  
[root@TE6410 /tmp]# ./thread-arm  
-/bin/sh: ./thread-arm: Permission denied  
[root@TE6410 /tmp]# chmod +x thread-arm  
[root@TE6410 /tmp]# ./thread-arm  
I am the thread  
I am the main-thread  
^C  
[root@TE6410 /tmp]#
```

10-4 实验四 多进程实验

1、实验目的:

- 了解什么是进程
- 练习多进程编程

2、实验内容:

多进程编程

3、实验设备：

PC 机 (Linux), 飞凌 6410 开发板

4、预备知识：

● 进程介绍

一个进程是一个 程序的一次执行的过程，程序是静态的，它是一些保存在磁盘上的可执行的代码和数据集合，进程是一个动态的概念。它是Linux 系统分配资源的基本单位。

Linux 进程中最知名的属性就是它的进程号 (Process Identity Number, PID) 和它的父进程号 (parent process ID, PPID)。PID、PPID 都是非零正整数。一个 PID 惟一地标识一个进程。一个进程创建新进程称为创建了子进程 (child process)。

● 进程在运行中的三种状态：

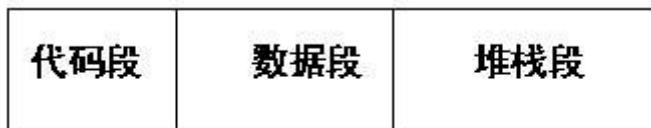
执行 (Running) 状态：CPU 正在执行，即进程正在占用 CPU。

就绪 (Waiting) 状态：进程已经具备的执行的一切条件，正在等待分配 CPU 的处理时间片。

停止 (Stoped) 状态：进程不能使用 CPU。

● 理解 Linux 下进程的结构

Linux 中一个进程在内存里有三部份的数据，就是“数据段”，“堆栈段”和“代码段”，基于 I386 兼容的中央处理器，都有上述三种段寄存器，以方便操作系统的运行。见图—1。



图—1 Linux 进程的结构

代码段，是存放了程序代码的数据，假如机器中有数个进程运行相同的一个程序，那么它们就可以使用同一个代码段。而数据段则存放程序的全局变量，常数以及动态数据分配的数据空间。

堆栈段存放的就是子程序的返回地址、子程序的参数以及程序的局部变量。堆栈段包括进程控制块 PCB (Process Control Block) 中。PCB 处于进程核心堆栈的底部，不需要额外分配空间。

● Linux 进程的创建

在 Linux 下产生新的进程的系统调用就是 fork 函数，这个函数名是英文中“分叉”的意思。为什么取这个名字呢？因为一个进程在运行中，如果使用了 fork，就产生了另一个进程，于是进程就“分叉”了，所以这个名字取得很形象。fork 的语法：

```
# include <unistd.h>  
  
pid_t pid;  
  
pid=fork();
```

调用 fork 时，系统将创建一个与当前进程相同的进程。他与原有的进程具有相同的数据、连接关系和在程序同一处执行的连续性。通常将原有的进程称为父进程，而把新生成的进程称为子进程。子进程是对父进程的复制，子进程获得同父进程相同的数据，但是同父进程使用不同的数据段和堆栈段。

Fork 调用将执行两次返回，从父子进程分别返回。如果 pid=0，则说明从子进程返回的，否则是从父进程返回的，此时返回的是子进程的 ID 号。可以在某一进程中调用 getpid() 函数来得到该进程的 ID 号。

● Linux 调度策略

1. SCHED_OTHER 分时调度策略，
2. SCHED_FIFO 实时调度策略，先到先服务
3. SCHED_RR 实时调度策略，时间片轮转

实时进程将得到优先调用，实时进程根据实时优先级决定调度权值，分时进程则通过 nice 和 counter 值决定权值，nice 越小，counter 越大，被调度的概率越大，也就是曾经使用了 cpu 最少的进程将会得到优先调度。

SCHED_RR 和 SCHED_FIFO 的不同：

当采用 SCHED_RR 策略的进程的时间片用完，系统将重新分配时间片，并置于就绪队列尾。放在队列尾保证了所有具有相同优先级的 RR 任务的调度公平。

SCHED_FIFO 一旦占用 cpu 则一直运行。一直运行直到有更高优先级任务到达或自己放弃。

如果有相同优先级的实时进程（根据优先级计算的调度权值是一样的）已经准备好，FIFO 时必须等待该进程主动放弃后才可以运行这个优先级相同的任务。而 RR 可以让每个任务都执行一段时间。

相同点：

RR 和 FIFO 都只用于实时任务。

创建时优先级大于 0 (1-99)。

按照可抢占优先级调度算法进行。

就绪态的实时任务立即抢占非实时任务。

所有任务都采用 Linux 分时调度策略时。

1. 创建任务指定采用分时调度策略，并指定优先级 nice 值 (-20~19)。

2. 将根据每个任务的 nice 值确定在 cpu 上的执行时间 (counter)。

3. 如果没有等待资源，则将该任务加入到就绪队列中。

4. 调度程序遍历就绪队列中的任务，通过对每个任务动态优先级的计算 (counter+20-nice) 结果，选择计算结果最大的一个去运行，当这个时间片用完后 (counter 减至 0) 或者主动放弃 cpu 时，该任务将被放在就绪队列末尾 (时间片用完) 或等待队列 (因等待资源而放弃 cpu) 中。

5. 此时调度程序重复上面计算过程，转到第 4 步。

6. 当调度程序发现所有就绪任务计算所得的权值都为不大于 0 时，重复第 2 步。

所有任务都采用 FIFO 时，

1. 创建进程时指定采用 FIFO，并设置实时优先级 rt_priority (1-99)。

2. 如果没有等待资源，则将该任务加入到就绪队列中。

3. 调度程序遍历就绪队列，根据实时优先级计算调度权值 (1000+rt_priority)，选择权值最高的任务使用 cpu，该 FIFO 任务将一直占有 cpu 直到有优先级更高的任务就绪 (即使优先级相同也不行) 或者主动放弃 (等待资源)。

4. 调度程序发现有优先级更高的任务到达(高优先级任务可能被中断或定时器任务唤醒，再或被当前运行的任务唤醒，等等)，则调度程序立即在当前任务堆栈中保存当前cpu寄存器的所有数据，重新从高优先级任务的堆栈中加载寄存器数据到cpu，此时高优先级的任务开始运行。重复第3步。

5. 如果当前任务因等待资源而主动放弃cpu使用权，则该任务将从就绪队列中删除，加入等待队列，此时重复第3步。

所有任务都采用RR调度策略时

1. 创建任务时指定调度参数为RR，并设置任务的实时优先级和nice值(nice值将会转换为该任务的时间片的长度)。
2. 如果没有等待资源，则将该任务加入到就绪队列中。
3. 调度程序遍历就绪队列，根据实时优先级计算调度权值($1000+rt_priority$)，选择权值最高的任务使用cpu。
4. 如果就绪队列中的RR任务时间片为0，则会根据nice值设置该任务的时间片，同时将该任务放入就绪队列的末尾。重复步骤3。
5. 当前任务由于等待资源而主动退出cpu，则其加入等待队列中。重复步骤3。

系统中既有分时调度，又有时间片轮转调度和先进先出调度

1. RR调度和FIFO调度的进程属于实时进程，以分时调度的进程是非实时进程。
2. 当实时进程准备就绪后，如果当前cpu正在运行非实时进程，则实时进程立即抢占非实时进程。
3. RR进程和FIFO进程都采用实时优先级做为调度的权值标准，RR是FIFO的一个延伸。FIFO时，如果两个进程的优先级一样，则这两个优先级一样的进程具体执行哪一个是由其在队列中的未知决定的，这样导致一些不公正性(优先级是一样的，为什么要让你一直运行?)，如果将两个优先级一样的任务的调度策略都设为RR，则保证了这两个任务可以循环执行，保证了公平。

● 进程的终止

在Linux环境中，一个进程的结束，可以调用相应的函数实现也可以是接收到某个信号而

结束。

```
1. # include <stdlib.h>
```

```
void exit(int status)
```

exit 函数是标准 C 中提供的函数，它用来终止正在运行的程序，他将关闭所有被该文件打开的文件描述符。

```
2. void abort(void)
```

调用 abort 函数将产生 SIGABRT 信号，该信号使进程非正常结束。

还可以通过 ps 查看进程的 ID 号，然后使用 kill 命令

如某进程的 pid=19056；则使用命令# kill 19056 来杀死 ID 号为 19056 的进程

调用 kill 命令时缺省产生的信号为 SIGTERM.

5 实验代码

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
main()
```

```
{
```

```
    pid_t pid;
```

```
    pid=fork();
```

```
    if(pid<0)
```

```
{
```

```
        printf("fork error!\n");
```

```
        return(1);
```

```
}
```

```
else if(pid==0)
```

```
{
```

```
    while(1)
```

```
{
```

```
    printf(" child process is running----PID=%d\n",getpid());
```

```
    sleep(1);  
}  
}  
else  
{  
    while(1)  
    {  
        printf("father process is running----PID=%d\n", getpid());  
        sleep(1);  
    }  
}  
return 0;  
}
```

6 编译及运行

A 本机编译并运行

gcc -o fork-pc fork.c

运行./ fork-pc

```
root@mdl-desktop:/usr/test/fork# gcc -o fork-pc fork.c  
root@mdl-desktop:/usr/test/fork# ./fork-pc  
child process is running----PID=6332  
father process is running----PID=6331  
child process is running----PID=6332  
father process is running----PID=6331  
child process is running----PID=6332  
father process is running----PID=6331  
^C  
root@mdl-desktop:/usr/test/fork#
```

B 交叉编译并运行

/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc -o fork-arm fork.c

将得到的可执行程序传送到开发板上运行。结果与在 PC 机上运行基本是一样的。

```
[root@TE6410 /tmp]# [root@TE6410 /tmp]#  
[root@TE6410 /tmp]# ls  
fork-arm  hello-arm  thread-arm  
[root@TE6410 /tmp]# chmod +x fork-arm  
[root@TE6410 /tmp]# ./fork-arm  
child process is running----PID=1145  
father process is running----PID=1144  
child process is running----PID=1145  
father process is running----PID=1144  
~C  
[root@TE6410 /tmp]#
```

10-5 实验五 网络编程实验—服务器/客户机

1、实验目的:

1. 了解 TCP/IP 协议
2. 掌握 socket 编程

2、实验内容:

实现典型客户机/服务器程序中的服务器及客户机

3、实验设备:

PC 机或 PC 机和飞凌 6410 开发板

4、预备知识:

- 计算机网络体系结构模式

所有的网络通信方式分为两种：线路交换和包交换。

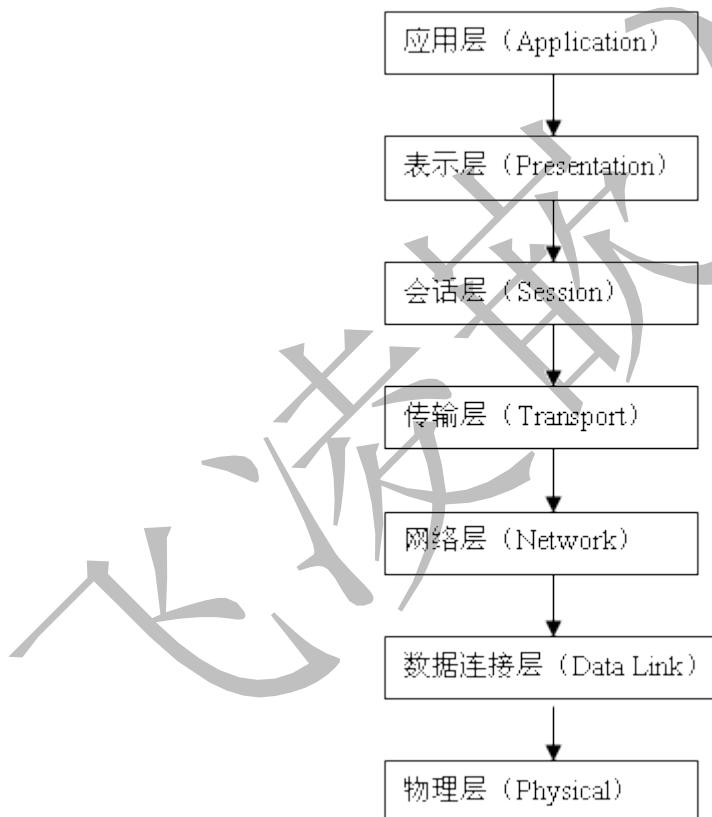
所谓的线路交换，就是指再传输时在发送端和接收端之间建立一个特定的线路连接，数据就可以在这条线路上传输。电话是采用的这种方式。

计算机网络则采用的是包交换，数据的发送端将要传输的数据分割成块，而每个块经过适当

的处理后形成一个数据包，包中有接收端的地址等必要信息，每个包单独传输。包中的数据并不是限定死的，只要保证数据的正确传输即可，具体应该定义哪些信息，则与使用的协议有关。

● OSI标准

OSI标准是开放系统互联标准（Open System Interconnection）即我们通常所说的网络互联的七层框架，他是1977年国际标准化组织提出的一种参考模型。值得注意的是，OSI并没有提供一个可以实现的方法，它不是一个标准而只是一个制定标准时使用的概念性的框架，更不是一个网络协议。



1、物理层（Physical Layer）：主要功能为定义了网络的物理结构，传输的电磁标准，Bit流的编码及网络的时间原则，如分时复用及分频复用。决定了网络连接类型（端到端或多端连接）及物理拓扑结构。说的通俗一些，这一层主要负责实际的信号传输。

- 2、链路层 (**Data Link Layer**): 在两个主机上建立数据链路连接，向物理层传输数据信号，并对信号进行处理使之无差错并合理的传输。
- 3、网络层 (**Network Layer**): 主要负责路由，选择合适的路径，进行阻塞控制等功能。
- 4、传输层 (**Transfer Layer**): 最关键的一层，向拥护提供可靠的端到端 (**End-to-End**) 服务，它屏蔽了下层的数据通信细节，让用户及应用程序不需要考虑实际的通信方法。
- 5、会话层 (**Session Layer**): 主要负责两个会话进程之间的通信，即两个会话层实体之间的信息交换，管理数据的交换。
- 6、表示层 (**Presentation Layer**): 处理通信信号的表示方法，进行不同的格式之间的翻译，并负责数据的加密解密，数据的压缩与恢复。
- 7、应用层 (**Application Layer**): 保持应用程序之间建立连接所需要的数据记录，为用户提供服务。

在工作中，每一层会给上一层传输来的数据加上一个信息头 (**header**)，然后向下层发出，然后通过物理介质传输到对方主机，对方主机每一层再对数据进行处理，把信息头去掉，最后还原成实际的数据。本质上，主机的通信是层与层之间的通信，而在物理上是从上向下最后通过物理信道到对方主机再从下向上传输。

● **TCP/IP协议**

在实际应用中，最重要的是TCP/IP (**Transport Control Protocol/Internet Protocol**) 协议，它是目前最流行的商业化的协议，相对于OSI，它是当前的工业标准或“事实的标准”，在1974年由Kahn提出的。它分为四个层次 (从高到低): 应用层 (与OSI的应用层对应)，传输层 (与OSI的传输层对应)，互联层 (与OSI的网络层对应)，主机-网络层 (与OSI的数据链路层和物理层对应)。

1. 应用层

应用层包括网络应用程序和网络进程，是与用户交互的界面，他为用户提供所需要的各种服务，包括远程登陆、文件传输和电子邮件等。他的作用相当于OSI中的应用层及表示层和会话层。

2. 传输层

相当于OSI中的传输层，他为应用程序提供通信服务，这种通信又叫端对端通信。他有三个主要协议：传输控制协议（TCP），用户数据包协议（UDP），和互联网控制消息协议（ICMP）。

TCP协议：以建立连接高可靠的传输为目的，他负责把大量的用户数据按一定的长度组成多个数据包进行发送，并在接收到数据包之后按分解顺序重组和恢复用户数据。他是一种面向连接的可靠的双向通信的数据流。

UDP协议：提供无连接数据包传输服务，他把用户数据分解为多个数据包后发送给接收方。它具有执行代码小，系统开销小和处理速度快等优点。

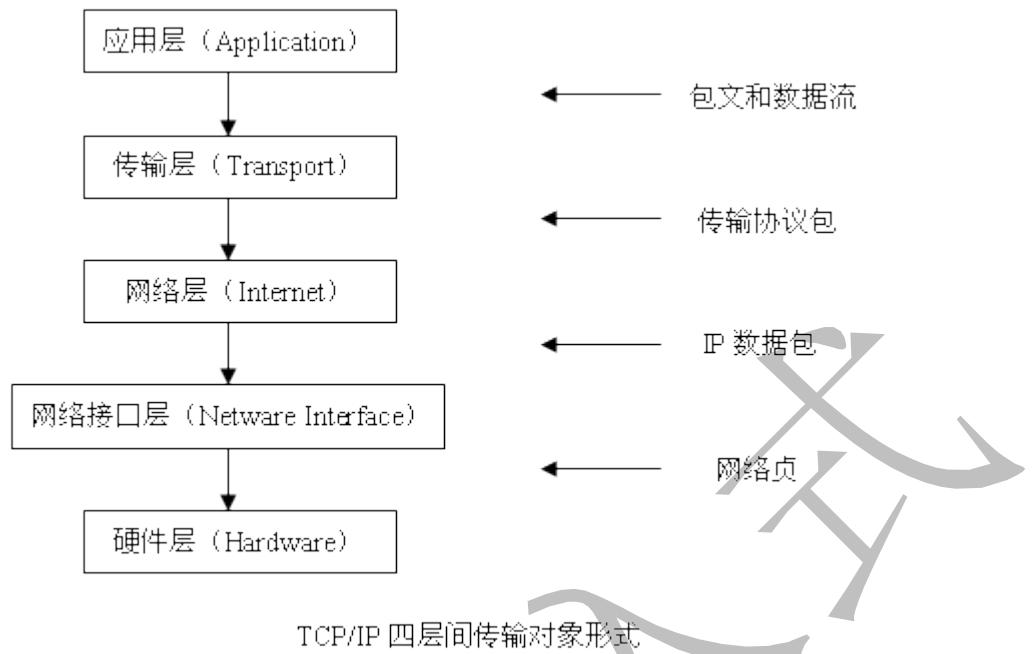
ICMP协议：主要用于端主机和网关以及互联网管理中心等地消息通信，以达到控制管理网络运行的目的。ICMP协议能发送出错消息给发送数据包的端主机，还有限制流量的功能。

3. 网络层

相当于osи的网络层，使用的协议是ip协议。他是用来处理机器之间的通信问题，他接受传输层请求，传输某个具有目的地址信息的分组。该层把分组封装到ip数据包中，填入数据包的头部（包头），使用路由算法来选择是直接把数据包发送到目标主机还是发给路由器，然后把数据包交给下面的网络接口层中的对应网络接口模块。

4. 网络接口层

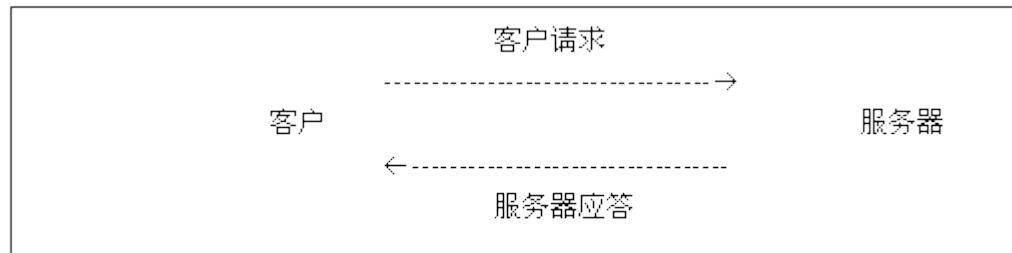
相当于osи中的数据连接层和物理层。他负责接收ip数据包和把数据包通过选定的网络发送出去,如图



● 客户机与服务器

TCP/IP 允许程序员在两个应用程序之间建立通信并来回传递数据，提供一种对等通信，这种对等应用程序可以在同一台机器上，也可以在不同的机器上运行。尽管 TCP/IP 指明了数据是如何在一对正在通信的应用程序间传递的，但是他并没有规定对等的应用程序在什么时间进行交互以及为什么要进行交互，也没有规定程序员在一个分布式环境下应该如何组织这些应用程序。实践中，有一种组织的方法在使用 TCP/IP 中占据着主要地位，现在网络上的绝大多数的通信应用程序都使用这种机制。

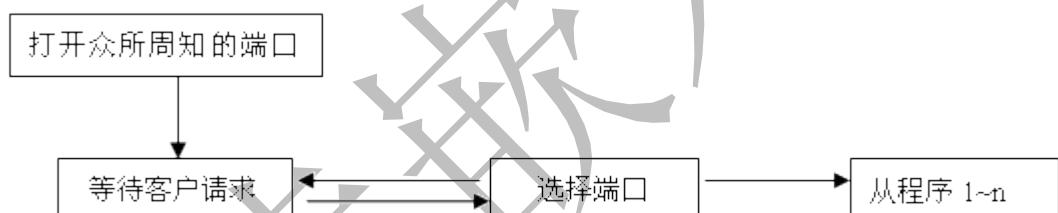
客户机/服务器模式要求每个应用程序应有两部分组成：一个部分负责启动通信，另一个部分负责对他进行应答。他们通常运行在不同的主机上，分别被称为客户机和服务器。服务器是指能在网络上可提供服务的任何程序；客户机是指用户为了得到某种服务所需要运行的应用程序。一个服务器接受网络上客户机的请求，完成服务后将结果返回给客户机，他们之间的关系如图：



服务器能完成简单和复杂的任务，以太主机可以同时运行多个服务器程序，一个服务器程序可以同时接收一个或多个客户的请求，当客户发送某个服务请求时，服务器使其在提供该服务器的端口排队，然后从队列中提取请求，为每个请求创建一个子进程，游子进程来处理具体的服务细节。

通常情况下，服务器包含两个部分：主程序和从程序。主程序负责接收来自客户的请求，从程序一般有几个，他们负责处理各个客户请求

主从程序工作过程：



如果客户请求所指的端口不是众所周知的端口，则应为它请求分配一个临时的端口，然后启动从程序，等待新的客户请求。从程序通常是一个子进程，处理完成一个客户请求后就终止并返回结果。

● 服务器软件设计的算法

最简单的服务器的算法：创建一个套接口，并将它绑定到以各种所周知的端口上，希望在这个端口上接受请求。然后就是一个无限迭代，在迭代中，服务器接收来自客户的下一个请求，处理这个请求，构造一个应答并发回给客户。但是实际中的设计问题更加复杂。

可以根据服务器在同一时刻处理的请求的个数将服务器分为并发服务器（concurrent server）和迭代服务器（iterative server）。迭代服务器是用来描述在一个时刻只处理一个客

户请求的服务器；而并发服务器是用来描述在一个时刻可以处理多个客户请求的服务器。以迭代方法实现的服务器易于构建和理解，而已并发方法实现的服务器虽然难于设计和构建，但是他拥有很好的性能。

还可以根据连接性问题把服务器分为面向连接的服务器和无连接的服务器，连接问题是传输协议的中心。在 TCP/IP 协议族中，TCP 提供的是一种面向连接传输服务，而 UDP 协议则提供的是一种无连接的服务。

1. 面向连接的服务器的算法

在面向连接的方法实现中，传输协议自动处理分组丢失和交付失序的问题，而服务器只要管理和使用这些连接就可以了，所以比较容易编程。算法如下：

- 服务器接收来自某个客户的入连接。
- 通过这个链接发送所有的通信数据。
- 从客户端接受请求并作出应答。
- 服务器在完成交互后关闭连接。

面向连接的设计要求对每个连接都有一个单独的套接口，而无连接的设计则允许从一个套接口上与多个主机通信。采用面向连接的设计时，在资源的使用上，服务器拥有分配给该连接的数据结构（包括缓冲区空间）的权利，并且这些资源不能被重新分配。因为服务器必须设计成始终在运行，所以不断有客户崩溃，服务器就可能因为耗尽资源而终止运行。

2. 无连接的服务器的算法

无连接的服务器没有资源耗尽问题的困扰，但是他不能依靠下层传输提供可靠的传递，这样通信的一方或双方就必须承担可靠支付的责任。如果用户要发送一个数据包，他可能到达，也可能中间丢失，更有传到时发生次序颠倒的可能，不过如果他到达了，那么报的内容是无错误的。

3. 迭代服务器的算法

- 创建一个套接口描述字，并绑定在众所周知的服务器端口上。
- 将该端口设置为被动模式，使其准备为一个服务器所用。
- 从套接口上接收下一个连接请求，获得该连接的一个新的套接字。
- 重复读取来自客户的请求，构造一个响应，按照应用协议向客户发回响应。
- 当与某个特定的用户完成交互时关闭连接，并返回第三步以接收一个新的连接。

4. 并发服务器的算法

大多数的并发服务器使用多进程来达到并发性，他们可以划分为主进程和从进程两类。一个主服务器进程最先开始执行，他在众所周知的端口上打开一个套接口，等待下一个请求，并为处理每个请求创建一个服务器进程，由一个从进程处理一个客户的通信。在从进程构成一个响应并将她发给客户后，从进程将退出。

并发的无连接服务器的算法采取如下的步骤。

主 1：创建一个 UDP 套接口并将其绑定到提供服务的众所周知的端口上。

主 2：重复调用 `recvfrom` 接受来自客户的下一个请求，创建一个新的从进程来处理响应。

从 1：由于接收到一个特定的请求以及访问到该套接口而被创建。

从 2：根据应用协议构造一个应答，并用 `sendto` 将该应答发回到客户。

从 3：从进程处理完一个请求后终止。

并发服务器使用面向连接协议的步骤：

主 1：创建一个 TCP 套接口，并将其绑定到提供服务的众所周知的端口上。

主 2：将该套接口设置为被动模式，作为服务器使用。

主 3：重复调用 `accept` 接受来自客户的下一个请求，并创建一个新的从进程来处理响应。

从 1：由于接受一个连接请求而被创建。

从 2：用该连接与客户进行交互，读取请求并发挥响应。

从 3：关闭链接并退出，在处理完来自一个客户的所有请求后，从进程就退出。因为创建进程比较昂贵，所以无连接服务器很少采用这种方法实现。

● 服务器死锁

服务器产生死锁的原因有很多，但导致的结果都一样，就是因为一个客户的行为而使服务器不能处理其他客户的请求，所以应该尽量避免。

● 套接口

套接口也就是网络进程的 ID。网络通信，归根到底是进程间的通信。在网络中，每一个节点都有一个网络地址，也就是 IP。网络地址只能确定进程所在的计算机，但无法确定是该计算机中的哪个进程，因此还需要其他信息也就是端口号（port）。在一台计算机中，一个端口号一

次智能分配给一个进程，也就是说，在一台计算机中，端口号和进程之间是一一对应的关系。所以，使用端口号和网络地址的组合就能唯一的确定整个网络中的一个网络进程。

把网络应用程序中所用到的网络地址和端口号信息放在一个结构体中，也就是套接口地址结构。每个协议族都定义它自己的套接口地址结构，套接口地址结构都以“`sockaddr_`”开头，并以每个协议族名中的两个字母作为结尾。

套接口根据协议不同可以分为很多类，主要就是 TCP 套接口（流式套接口）和 UDP 套接口（数据包套接口）。

- **TCP 协议建立连接时使用“三段握手 TWH”方式：**

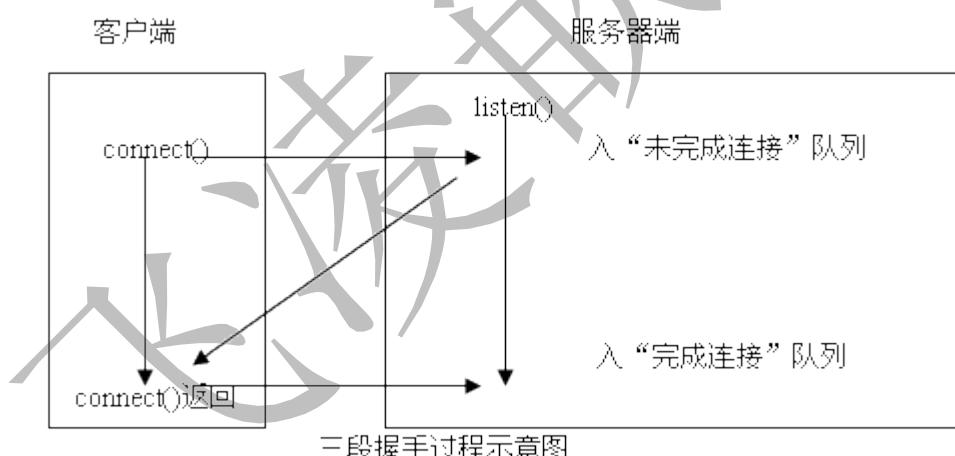
客户端先用 `connect()` 向服务器发出一个要求连接的信号 `SYN1`。

服务器进程接收到这个信号后，发回应答信号 `ack1`，同时这也一个要求回答的信号 `SYN2`。

客户端收到信号 `ack1` 和 `SYN2` 后再次应答 `ack2`。

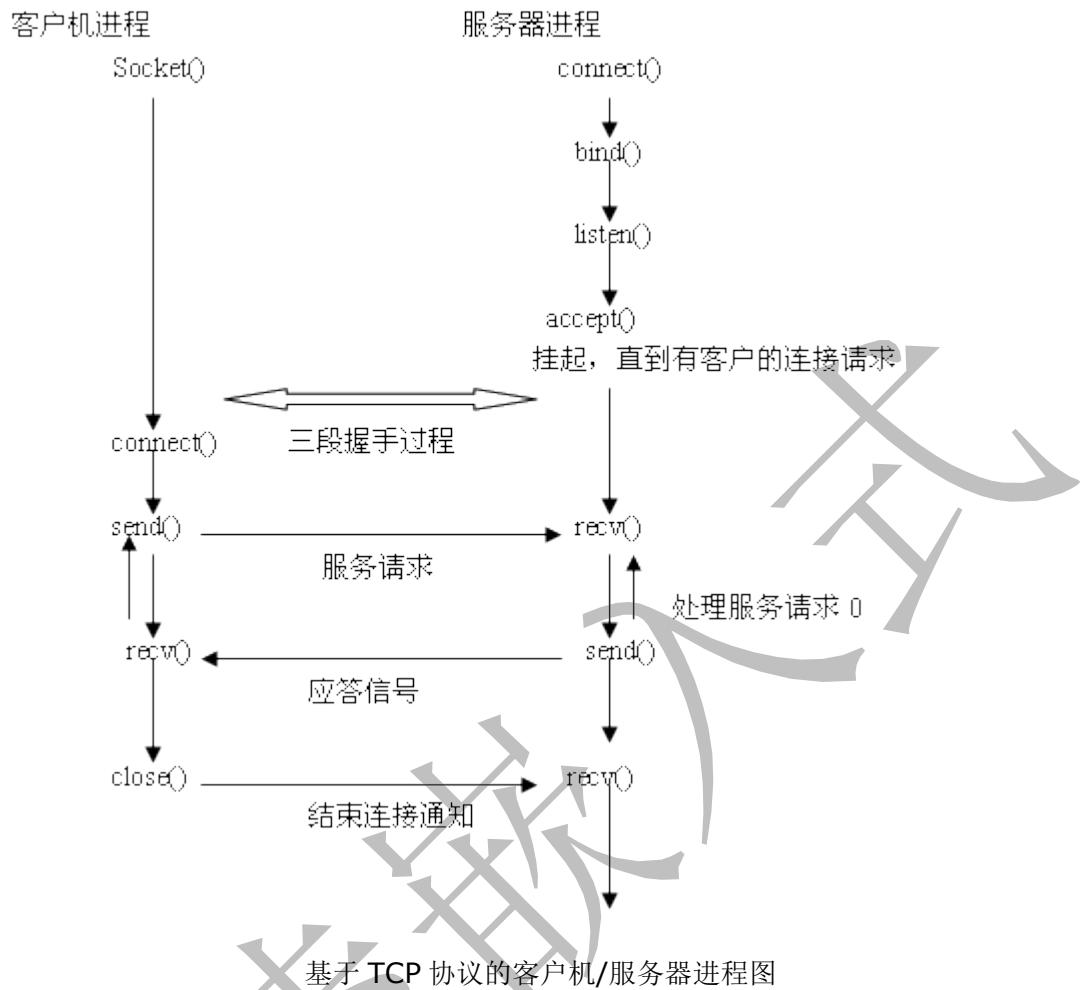
服务器收到应答信号 `ack2`，一次连接才算建立完成。

具体过程如下图：



- **基于 TCP 的客户机/服务器**

使用 TCP 协议的客户机/服务器进程的工作过程如下图：



● Socket 编程相关函数

socket, bind, listen, accept , connect

(1) socket(建立连接)

表头文件 #include<sys/socket.h>

定义函数: int socket(int family,int type,int protocol);

函数说明: socket()函数用来生成一个套接口描述字，也称为套接字，指定协议族和套接口

参数: family指定协议族, type指明字节流方式, 而protocol一般为0

Family的取值范围:

AF_LOCAL	UNIX协议族
AF_ROUTE	路由套接口
AF_INET	IPv4协议
AF_INET6	IPv6协议
AF_KEY	密钥套接口

参数type的取值范围:

SOCK_STREAM	TCP套接口
SOCK_DGRAM	UDP套接口
SOCK_PACKET	支持数据链路访问
SOCK_RAM	原始套接口

返回值: 成功返回非负描述字, 失败返回负值

(2) bind (对socket定位)

表头文件 `#include<sys/types.h>`
`#include<sys/socket.h>`

定义函数 `int bind(int sockfd,struct sockaddr * my_addr,int addrlen);`

函数说明 `bind()`用来设置给参数`sockfd`的socket一个名称。此名称由参数`my_addr`指向一个`sockaddr`结构, 对于不同的socket domain定义了一个通用的数据结构

`struct sockaddr`

{

`unsigned short int sa_family;`

`char sa_data[14];`

}

`sa_family` 为调用`socket ()`时的domain参数, 即`AF_xxxx`值。

`sa_data` 最多使用14个字符长度。

此`sockaddr`结构会因使用不同的socket domain而有不同结构定义，例如使用`AF_INET` domain，其`socketaddr`结构定义便为

```
struct socketaddr_in
{
    unsigned short int sin_family;
    uint16_t sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[8];
};
```

struct in_addr

```
{
```

 uint32_t s_addr;

};

`sin_family` 即为`sa_family`

`sin_port` 为使用的port编号

`sin_addr.s_addr` 为IP 地址

`sin_zero` 未使用。

参数 `socket`为套接字，`my_addr`是一个指向特定协议地址结构的指针，`addrlen`为`sockaddr`的结构长度。

返回值 成功则返回0，失败返回-1，错误原因存于`errno`中。

错误代码

`EBADF` 参数`sockfd` 非合法socket处理代码。

`EACCESS` 权限不足

`ENOTSOCK` 参数`sockfd`为一文件描述词，非socket。

(3) listen (等待连接)

相关函数 socket, bind, accept, connect

表头文件 #include<sys/socket.h>

定义函数 int listen(int s,int backlog);

函数说明 listen()用来等待参数s 的socket连线。参数backlog指定同时能处理的最大连接要求，如果连接数目达此上限则client端将收到ECONNREFUSED的错误。Listen()并未开始接收连线，只是设置socket为listen模式，真正接收client端连线的是accept()。通常listen()会在socket(), bind()之后调用，接着才调用accept()。

返回值 成功则返回0，失败返回-1，错误原因存于errno

附加说明 listen()只适用SOCK_STREAM或SOCK_SEQPACKET的socket类型。如果socket为AF_INET则参数backlog 最大值可设至128。

错误代码

EBADF 参数sockfd非合法socket处理代码

EACCES 权限不足

EOPNOTSUPP 指定的socket并未支援listen模式。

(4) accept

表头文件 #include<sys/types.h>

#include<sys/socket.h>

定义函数 int accept(int s, struct sockaddr * addr, int * addrlen);

函数说明 accept()用来接受参数s的socket连线。参数s的socket必需先经bind()、listen()函数处理过，当有连线进来时accept()会返回一个新的socket处理代码，往后的数据传送与读取就是经由新的socket处理，而原来参数s的socket能继续使用accept()来接受新的连线要求。连线成功时，参数addr所指的结构会被系统填入远程主机的地址数据，参数addrlen为sockaddr的结构长度。关于结构sockaddr的定义请参考bind()。

返回值 成功则返回新的socket处理代码，失败返回-1，错误原因存于errno中。

错误代码 EBADF 参数s 非合法socket处理代码。

EFAULT 参数addr指针指向无法存取的内存空间。

ENOTSOCK 参数s为一文件描述词，非socket。

EOPNOTSUPP 指定的socket并非SOCK_STREAM。

EPERM 防火墙拒绝此连线。

ENOBUFS 系统的缓冲内存不足。

ENOMEM 核心内存不足。

(5) connect (建立socket连线)

相关函数 socket, bind, listen

表头文件 #include<sys/types.h>

#include<sys/socket.h>

定义函数 int connect (int sockfd, struct sockaddr * serv_addr, int addrlen);

函数说明 connect()用来将参数sockfd 的socket 连至参数serv_addr 指定的网络地址。结构sockaddr请参考bind()。参数addrlen为sockaddr的结构长度。

返回值 成功则返回0，失败返回-1，错误原因存于errno中。

错误代码 EBADF 参数sockfd 非合法socket处理代码

EFAULT 参数serv_addr指针指向无法存取的内存空间

ENOTSOCK 参数sockfd为一文件描述词，非socket。

EISCONN 参数sockfd的socket已是连线状态

ECONNREFUSED 连线要求被server端拒绝。

ETIMEDOUT 企图连线的操作超过限定时间仍未有响应。

ENETUNREACH 无法传送数据包至指定的主机。

EAFNOSUPPORT sockaddr结构的sa_family不正确。

EALREADY socket 为不可阻断且先前的连线操作还未完成。

(6) send()和recv()这两个函数用于面向连接的socket上进行数据传输。

函数定义: int send(int sockfd, const void *msg, int len, int flags);

参数: Sockfd是你想用来传输数据的socket描述符; msg是一个指向要发送数据的指针; Len是以字节为单位的数据的长度; flags一般情况下置为0 (关于该参数的用法可参照man手册)。

返回值: send()函数返回实际上发送出的字节数, 可能会少于你希望发送的数据。在程序中应该将send()的返回值与欲发送的字节数进行比较。当send()返回值与len不匹配时, 应该对这种情况进行处理。

```
char *msg = "Hello!";
int len, bytes_sent;
.....
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.....
```

函数定义: int recv(int sockfd,void *buf,int len,unsigned int flags);

参数: sockfd是接受数据的socket描述符; buf 是存放接收数据的缓冲区; len是缓冲的长度。

Flags也被置为0。

返回值：Recv()返回实际上接收的字节数，当出现错误时，返回-1并置相应的errno值。

sendto()和recvfrom()用于在无连接的数据报socket方式下进行数据传输。由于本地socket并没有与远端机器建立连接，所以在发送数据时应指明目的地址。

5 实验代码

```
*****client.c*****
*****
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>

int main(int argc,char **argv)
{
    pid_t fd;
    const char buff[] = "Hello! Welcome here ggggggggggg!\r\n"; //定义要发送的数据缓冲区;
    int sockfd,connsock; //定义一个socket套接字, 用于通讯
    struct sockaddr_in serveraddr;//定义网络套接字地址结构
    if(argc!= 2)
    {
        printf("Usage: echo ip地址");
        exit(0);
    }
    sockfd =socket(AF_INET, SOCK_STREAM, 0); //创建一个套接字
    bzero(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET; //指定使用的通讯协议族
    serveraddr.sin_port = htons(5000); //指定要连接的服务器的端口
    inet_pton(AF_INET, argv[1], &serveraddr.sin_addr);
```

```
connect(sockfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)); //连接服务器
send(sockfd, buff, sizeof(buff), 0); //向客户端发送数据
close(sockfd); //关闭套接字
return(0);
}

/*********************service.c******/
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>

void main()
{
    pid_t fd;
    int listensock, connsock;
    char recvbuff[100]; //定义要接收的数据缓冲区
    struct sockaddr_in serveraddr; //定义网络套接字地址结构
    listensock = socket(AF_INET, SOCK_STREAM, 0); //创建一个套接字，用于监听
    bzero(&serveraddr, sizeof(struct sockaddr)); //地址结构清零
    serveraddr.sin_family = AF_INET; //指定使用的通讯协议族
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY); //指定接受任何连接
    serveraddr.sin_port = htons(5000); //指定监听的端口
    bind(listensock, (struct sockaddr *)&serveraddr, sizeof(struct sockaddr_in)); //给套接
    口邦定地址
    listen(listensock, 1024); //开始监听
    connsock = accept(listensock, (struct sockaddr *)NULL, NULL);
    //建立通讯的套接字，accept函数，等待客户端程序使用connect函数的连接
    recv(connsock, recvbuff, sizeof(recvbuff), 0); //接收服务器的数据
    printf("%s\n", recvbuff); //打印接收到的数据
    sleep(2);
```

```

        close(connsock); //关闭通讯套接字
        close(listensock); //关闭监听套接字
    }
}

```

6、编译及运行

本实验是运行在PC和开发板间，通过网口通讯，所以两部分代码谁做客户端谁做服务器没有明确要求，基本取决于编译方式。下面给出的编译方式决定开发板做了服务器，PC做了客户端。

```

root@mdl-desktop:/usr/test/socket# /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-g
cc -o se service.c
root@mdl-desktop:/usr/test/socket# gcc client.c -o cl
root@mdl-desktop:/usr/test/socket# ls
cl  client.c  se  service.c
root@mdl-desktop:/usr/test/socket#

```

把se传送到开发板上，运行

```

[root@TE6410 /tmp]# [root@TE6410 /tmp]#
[root@TE6410 /tmp]# ls
se
[root@TE6410 /tmp]# chmod +x se
[root@TE6410 /tmp]# ./se
-
```

在PC机上运行可执行程序cl，命令如下：

./cl 192.168.0.232 //192.168.0.232是服务器IP，此处是开发板IP。此命令是客户端要与IP为192.168.0.232的服务器相连。

```

root@mdl-desktop:/usr/test/socket# ./cl 192.168.0.232
root@mdl-desktop:/usr/test/socket#

```

客户端运行该命令之后，看一下服务器端有什么现象

```

[root@TE6410 /tmp]# [root@TE6410 /tmp]#
[root@TE6410 /tmp]# ls
se
[root@TE6410 /tmp]# chmod +x se
[root@TE6410 /tmp]# ./se
Hello! Welcome here  ggggggggggggg!
[root@TE6410 /tmp]#

```

10-6 实验六 Makefile 实验

1、实验目的：

- 了解 make

- 了解 makefile
- 掌握 makefile 编程

2、实验内容：

通过 makefile 来编译程序

3、实验设备：

PC 机、飞凌 6410 开发板

4、预备知识：

● Make

Make 是大型程序的维护工具，当你有一个 c 文件时，你可以用 gcc 直接编译，但当你的主程序依赖于很多其他 c 文件时，你再用 gcc 一条一条的编译便显得很吃力，这时候用 make 便会使一切变得轻松。那 make 是如何维护的呢？这就需要 makefile。

● makefile

当运行 make 命令时，make 命令会在当前目录下按顺序寻找文件名为“GUNmakefile”、“Makefile”、“makefile”的文件，找到后解释这些文件。所以说 make 是一个解释 makefile 中指令的命令工具。

Makefile 或 makefile：告诉 make 维护一个大型程序，该做什么。Makefile 说明了组成程序的各模块间的相互关系及更新模块时必须进行的动作，make 按照这些说明自动地维护这些模块。

简单的说 makefile 就像批处理的脚本文件，里边写好了一些命令的集合，当运行 make 命令时，便会接着 makefile 提供的命令及顺序来完成编译。

Makefile 文件包含了五部分内容：显示规则、隐式规则、变量定义、文件指示和注释

✓ **显示规则：**显示规则说明了如何生成一个或多个目标文件。这要由 makefile 文件的创作者指出，包括要生成的文件、文件的依赖文件、生成的命令。

- ✓ **隐式规则：**由于 **makefile** 有自动推导功能，所以隐式的规则可以比较粗糙地简略书写 **makefile** 文件，这是由 **make** 所支持的。
- ✓ **变量定义：**在 **makefile** 文件中要定义一系列的变量，变量一般都是字符串，像 C 语言中的宏。当 **makefile** 文件被执行时，其中的变量都会扩展到相应的引用位置上。
- ✓ **文件指示：**包括 3 个部分，一个是在一个 **makefile** 文件中引用另一个 **makefile** 文件，就像 C 语言中的 **include** 一样；另一个是指根据某些情况指定 **makefile** 文件中有效部分，就像 C 语言中的预编译**#if** 一样；还有就是定义一个多行的命令。
- ✓ **注释部分：****makefile** 文件中只有行注释，和 UNIX 的 **shell** 脚本一样，其注释用“#”字符，就像 C 语言中的“/* */”、“//”一样。如果要在 **makefile** 文件中使用“#”字符，可以用反斜杠进行转义如“#”。

● **Makefile** 中的变量

Makefile 里的变量就像一个环境变量。事实上，环境变量在 **make** 中也被解释成 **make** 的变量。这些变量对大小写敏感，一般使用大写字母。几乎可以从任何地方引用定义的变量。

Makefile 中的变量是用一个文本串在 **Makefile** 中定义的，这个文本串就是变量的值。只要在一行的开始写下这个变量的名字，后面跟一个“=”号，以及要设定这个变量的值即可定义变量，下面是定义变量的语法：

VARNAME=string

使用时，把变量用括号括起来，并在前面加上\$符号，就可以引用变量的值：

\$(VARNAME)

其中 **GNUMake** 种主要预定义的变量：

\$* 不包含扩展名的目标文件名称。

\$+ 所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件。

\$< 第一个依赖文件的名称。

\$? 所有的依赖文件，以空格分开，这些依赖文件的修改日期比目标的创建日期晚。

\$@ 目标的完整名称。

\$^ 所有的依赖文件，以空格分开，不包含重复的依赖文件。

\$% 如果目标是归档成员，则该变量表示目标的归档成员名称。例如，如果目标名称为

mytarget.so(image.o), 则 \$@ 为 mytarget.so, 而 \$% 为 image.o。

● Make工作时的执行步骤

- 读入所有的makefile文件
- 读入被include包括的其他的makefile文件
- 初始化文件中的变量
- 推到隐式规则，并分析所有规则
- 为所有的目标文件创建依赖关键链
- 根据依赖关系，决定哪些目标要重新生成
- 执行生成命令

第(1)~(5)步为第一个阶段，第(6)~(7)步为第二个阶段。第一个阶段中，如果定义的变量被使用了，make会在它使用的位置把它展开。但make并不会马上完全展开，make使用的是拖延战术。如果变量出现在依赖关系的规则中，进当这条依赖关系决定要使用时，变量才会在其内部展开。

以下边的Makefile为例，解释一下make是怎么运行的：

Makefile 文件中定义的第一个目标，make首先将其读入，然后从第一行开始执行，把第一个目标 test 作为它的最终目标，所有后面的目标的更新都会影响到 test 的更新。第一条规则说明只要文件 test 的时间戳比文件 prog.o 或 code.o、main.o 中的任何一个旧，下一行的编译命令将会被执行。在检查文件 prog.o 和 code.o、main.o 的时间戳之前，make会在下面的行中寻找以 prog.o 和 code.o、main.o 为目标的规则，在第五行中找到了关于 prog.o 的规则，该文件的依赖文件是 prog.c、prog.h 和 code.h。同样，make会在后面的规则行中继续查找这些依赖文件的规则，如果找不到，则开始检查这些依赖文件的时间戳，如果这些文件中任何一个的时间戳比 prog.o 的新，make 将执行 “gcc -c prog.c -o prog.o” 命令，更新 prog.o 文件。以同样的方法，接下来对文件 code.o、main.o 以同样的方法，接下来对文件 code.o、main.o 做类似的方法。当 make 执行完所有这些嵌套的规则后，make 将处理最顶层的 test 规则。如果关于 prog.o 和 code.o、main.o 的三个规则中的任何一个被执行，至少其中一个.o 目标文件就会比 test 新，那么就要执行 test 规则中的命令，因此 make 去执行 gcc 命令将 prog.o 和 code.o、main.o 连接成目标文件 test。

5、实验代码

```
*****code.c*****
*****
#include "code.h"
```

```
#include <stdio.h>
void circle(float r)
{
    printf(format_circle, 2*PI*r);
}

/*********************prog. c********************/
#include "prog.h"
#include "code.h"
#include <stdio.h>
area(float r)
{
    printf("r=%f\n", r);
    printf(format_area, PI*r*r);
    printf("thank you\n");
}
/*********************test. c********************/
#include <stdio.h>
extern area(float);
extern circle(float);
main()
{
    printf("dgjdkfhgkjfdg\n");
    area(2.5);
    circle(2.5);
    return 0;
}
/*********************code. h********************/
#define PI 3.1415926
#define format_circle "circle=%f\n"
/*********************prog. h********************/
#define format_area "area=%f\n"

/*********************Makefile********************/
CC=/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc
#CC=gcc
test: prog.o code.o main.o
    $(CC) -o test prog.o code.o main.o
main.o: test.c
    $(CC) -o main.o -c test.c
prog.o: prog.c prog.h code.h
```

```
$(CC) -c prog.c -o prog.o  
code.o: code.c code.h  
    $(CC) -c code.c -o code.o  
clean:  
    rm -f *.o
```

6、编译及运行

A、本机编译

本机编译需要使用本机编译器，修改Makefile文件：也就是将CC改成gcc“CC=gcc”，将交叉编译器屏蔽掉（用#屏蔽）

```
#CC=/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc
```

```
CC=gcc
```

保存退出

编译的时候直接在命令行敲“make”，最后生成可执行文件test，运行如下

```
root@mdl-desktop:/usr/test/makefile# make  
gcc -c prog.c -o prog.o  
gcc -c code.c -o code.o  
gcc -o main.o -c test.c  
gcc -o test prog.o code.o main.o  
root@mdl-desktop:/usr/test/makefile# ./test  
make file test start :  
r=2.500000  
area=19.634954  
thank you!  
circle=15.707963  
root@mdl-desktop:/usr/test/makefile#
```

B、交叉编译

修改Makefile文件，屏蔽掉Makefile文件中的#CC=gcc(用#屏蔽)，这样使编译器指向arm-linux-gcc
CC=/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc

```
#CC=gcc
```

保存退出

编译的时候直接在命令行敲“make”，最后生成可执行文件test，运行如下

```
[root@TE6410 /tmp]# [root@TE6410 /tmp]#  
[root@TE6410 /tmp]# ls  
test  
[root@TE6410 /tmp]# chmod +x test  
[root@TE6410 /tmp]# ls  
test  
[root@TE6410 /tmp]# ./test  
make file test start :  
r=2.500000  
area=19.634954  
thank you!  
circle=15.707963  
[root@TE6410 /tmp]#
```

10-7 实验七 进程间通讯

1、实验目的:

- 了解进程间通信原理
- 掌握进程间通信方式

2、实验内容:

编程实现信号量和共享内存

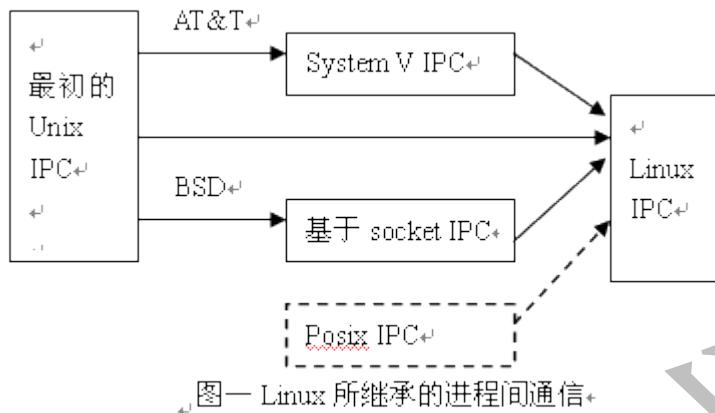
3、实验设备:

PC机或飞凌6410开发板

4、预备知识:

- 基本介绍

Linux 下的进程通信手段基本上是从 Unix 平台上的进程通信手段继承而来的。而对 Unix 发展做出重大贡献的两大主力 AT&T 的贝尔实验室及 BSD (加州大学伯克利分校的伯克利软件发布中心) 在进程间通信方面的侧重点有所不同。前者对 Unix 早期的进程间通信手段进行了系统的改进和扩充, 形成了“system V IPC”, 通信进程局限在单个计算机内; 后者则跳过了该限制, 形成了基于套接口 (socket) 的进程间通信机制。Linux 则把两者继承了下来, 如图示:



其中，最初 Unix IPC 包括：管道、FIFO、信号；System V IPC 包括：System V 消息队列、System V 信号灯、System V 共享内存区；Posix IPC 包括：Posix 消息队列、Posix 信号灯、Posix 共享内存区。有两点需要简单说明一下：1) 由于 Unix 版本的多样性，电子电气工程协会（IEEE）开发了一个独立的 Unix 标准，这个新的 ANSI Unix 标准被称为计算机环境的可移植性操作系统界面（POSIX）。现有大部分 Unix 和流行版本都是遵循 POSIX 标准的，而 Linux 从一开始就遵循 POSIX 标准；2) BSD 并不是没有涉足单机内的进程间通信（socket 本身就可以用于单机内的进程间通信）。事实上，很多 Unix 版本的单机 IPC 留有 BSD 的痕迹，如 4.4BSD 支持的匿名内存映射、4.3+BSD 对可靠信号语义的实现等等。

图一给出了 Linux 所支持的各种 IPC 手段，在本文接下来的讨论中，为了避免概念上的混淆，在尽可能少提及 Unix 的各个版本的情况下，所有问题的讨论最终都会归结到 Linux 环境下的进程间通信上来。并且，对于 Linux 所支持通信手段的不同实现版本（如对于共享内存来说，有 Posix 共享内存区以及 System V 共享内存区两个实现版本），将主要介绍 Posix API。

● Linux 下进程间通信的几种主要手段简介：

- ◆ 管道（Pipe）及有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；
- ◆ 信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；Linux 除了支持 Unix 早期信号语义函数 `sigal` 外，还支持语义符合 Posix.1 标准的信号函数 `sigaction`（实际上，

该函数是基于 BSD 的, BSD 为了实现可靠信号机制, 又能够统一对外接口, 用 `sigaction` 函数重新实现了 `signal` 函数);

- ◆ 报文 (Message) 队列 (消息队列): 消息队列是消息的链接表, 包括 Posix 消息队列 system V 消息队列。有足够的权限的进程可以向队列中添加消息, 被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少, 管道只能承载无格式字节流以及缓冲区大小受限等缺点。
- ◆ 共享内存: 使得多个进程可以访问同一块内存空间, 是最快的可用 IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制, 如信号量结合使用, 来达到进程间的同步及互斥。
- ◆ 信号量 (semaphore): 主要作为进程间以及同一进程不同线程之间的同步手段。
- ◆ 套接口 (Socket): 更为一般的进程间通信机制, 可用于不同机器之间的进程间通信。

起初是由 Unix 系统的 BSD 分支开发出来的, 但现在一般可以移植到其它类 Unix 系统上: Linux 和 System V 的变种都支持套接字。

基于上述几种方式, 我们主要讨论共享内存和信号量

● 共享内存

共享内存是 Linux 系统中最底层的通讯机制, 也是最快速的通信机制。两个不同进程 A、B 共享内存的意思是, 同一块物理内存被映射到进程 A、B 各自的进程地址空间。进程 A 可以即时看到进程 B 对共享内存中数据的更新, 反之亦然。由于多个进程共享同一块内存区域, 必然需要某种同步机制, 互斥锁和信号量都可以。

使用共享内存有两种方法: 映射/`/dev/mem` 设备和内存映像文件。比较而言, 内存映像文件比较安全, 使用映射/`/dev/mem` 设备可能引起系统崩溃, 但内存映像文件会带来文件系统的额外开销。我们主要讨论映射/`/dev/mem` 设备实现共享内存存的方法。

每个共享内存都有一个与其相对应的结构 `shmid_ds`, 该结构定义如下:

```
struct shmid_ds
{
    struct ipc_perm shm_perm; //对于该共享内存的是 ipc_perm 结构指针
    int shm_segsz;           //以字节表示的共享内存区域的大小
```

```

ushort shm_lkcnt;           //共享内存区域被锁定的时间数

pid_t shm_cpid;            // 创建该共享内存的进程 ID

pid_t shm_lpid;             //最后一次调用 shmem 函数的进程 ID

ulong shm_nattch;           //当前使用该共享内存区域的进程数

time_t shm_atime;           //最近一次附加操作的时间

time_t shm_dtime;           //最近一次分离操作的时间

time_t shm_ctime;           //最近一次改变的时间

};


```

(1) 共享内存创建与打开

```

#include <sys/types.h>
#include <sys/IPC.h>
#include <sys/shm.h>
int shmget(key_t key , int size ,int flag);

```

参数说明：

key 表示所创建或打开的共享内存的键

size 表示共享内存区域的大小，指再创建一个新的共享内存时生效。

flag 表示调用函数的操作类型，也可用于设置共享内存的访问权限，两者通过逻辑或表示。

shmget()不仅可以创建一个新的共享内存，也可以用于打开一个已存在的共享内存。

调用该函数的作用由参数 **key** 和 **flag** 决定，相应约定如下：

1. 当 **key** 为 **IPC_PRIVATE** 时，创建一个新的共享内存，此时参数 **flag** 的取值对函数的操作不起任何作用。
2. 当 **key** 不为 **IPC_PRIVATE** 时，且 **flag** 设置了 **IPC_CREAT** 位，而没有设置 **IPC_EXCL** 位，则执行操作有 **key** 取值决定。如果 **key** 为内核中某个已存在的共享内存的键，则执行打开这个键的操作；反之则执行创建共享内存的操作。
3. 当 **key** 不为 **IPC_PRIVATE** 时，且 **flag** 同时设置了 **IPC_CREAT** 位和 **IPC_EXCL** 位，则只执行创建共享内存的操作。参数 **key** 的取值应与内核中已存在的任何共享内存的

键值都不相同，否则函数调用失败。

此函数调用成功时，返回值为共享内存的引用标示符；调用失败时，返回-1。当调用该函数创建一个共享内存时，此共享内存的 **shmid_ds** 结构被初始化。**Ipc_perm** 中的各个域被设置为相应值，**shm_lpid**、**shm_nattch**、**shm_atime** 和 **shm_dtime** 被设置为 0，**shm_ctime** 设置为当前时间。

(2) 附加

当一个共享内存创建或被打开后，某个进程如果要使用该共享内存则必须将此内存区域附加到他的地址空间，相关函数如下：

```
#include <sys/type.h>
#include <sys/ipc.h>
#include <sys/shm.h>
void *shmat (int shmid , void * addr , int flag)
```

参数说明：

Shmid 表示要附加的共享内存段的引用表示符

Flag 表示 **shmat** 函数的操作方式，如果 **flag** 设置了 **SHM_RDONLY** 位，该内存区域被设置为只读，否则设置为可读写。

Addr 和 **flag** 共同决定共享内存区域要放附加到的地址值，相应约定如下：

如果 **addr** 为 0，系统将自动查找进程地址空间，将共享内存区域附加到第一块有效内存区域上，此时 **flag** 无效。

如过 **addr** 不为 0，而 **flag** 未设置 **SHM_RND** 位，则共享内存附加到由 **addr** 指定的地指处。

如过 **addr** 不为 0，而 **flag** 设置 **SHM_RND** 位，则共享内存附加到由 **addr_(addrmodSHMLBA)**指定的地指处。

此函数调用成功时，返回共享内存区域的指针；调用失败时，返回值为-1.

(3) 分离

当一个进程对共享内存区域的访问完成后，可以调用 **shmdt** 函数使共享内存区域与该进程的地址空间分离，相关函数如下：

```
#include <sys/type.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt (void *addr);
```

此函数仅用于将共享内存区域与进程的地址空间分离，并不删除共享内存本身。参数 **addr** 为要分离的共享内存区域的指针，是调用 **shmat** 函数的返回值。调用成功时，返回值为 0；失败时返回-1。

(4) 共享内存控制

对共享内存区域的具体控制操作是通过函数 **shmctl** 来实现的，说明如下：

```
#include <sys/type.h>
#include <sys/ipc.h>
#include <sys/shm.h>
Int shmctl (int shmid , int cmd , shmid_ds *buf);
```

参数说明：

Shmid 为共享内存的引用标示符

Buf 是指向 **shmid_ds** 结构体的指针。

Cmd 表示调用该函数希望执行的操作，其取值和对应操作如下：

SHM_LOCK: 将共享内存区域上锁，只能由超级用户执行

IPC_RMID: 用于删除共享内存。执行该命令时，共享内存的引用标示符立刻被删除，则该共享内存不能在被其他进程所附加。但共享内存的真正删除要到所有附加了该共享内存的进程结束或断开于该共享内存的连接时才执行。

IPC_SET: 按参数 **buf** 指向的结构中的值设置该共享内存对应的 **shmid_ds** 结构。只有有效用户 ID 和共享内存的所有者 ID 或创建者 ID 相同的用户进程，以及超级用户进程可移植性这一操作。

IPC_STAT: 用于取得该共享内存的是 **shmid_ds** 结构，保存于 **buf** 指向的缓冲区。

SHM_UNLOCK: 将上锁的共享内存区域释放，只能由超级用户执行。

● 信号量

信号量是一种用于多个进程访问共享资源进行控制的机制。共享资源通常可分为两大类：一类是互斥共享资源，即任一时刻只允许一个进程访问该资源；一类是同步共享资源，同一是可允许多个进程访问该资源。

信号量是一个计数器的值，它可以被几个进程作为一个集合义原子的方式执行。信号量的计数器控制着对资源的访问控制，信号量提供了两个主要的操作来处理计数器的值：

- (1) 资源的使用者在使用资源之前等待信号量。如果信号量的值为 0，则继续等待，如果大于 0，则将信号量值减 1，使用者开始使用资源。
- (2) 资源的使用者在资源使用完毕后通知信号量。使用者通知信号量不再使用资源了，信号量的值加 1，检查等待信号量的使用者的序列，以确定是否有其他的使用者在等待之中。

实际上，SYSV 子系统提供的信号量机制要复杂的多。不能单独定义一个信号量，而只能定义一个信号量集，其中包括一组信号量，统一信号量集中的信号量使用同一引用 ID，这样设置是为了多个资源或同步操作的需要。每个信号量集都有一个与其相对应的结构，其中记录了信号量集的各种信息，该结构定义如下：

```
struct semid_ds
{
    struct ipc_perm sem_perm; // 对应于该信号量集的 ipc_perm 结构指针
    struct sem *sem_base; // 指向信号量集中第一个信号量的 sem 结构
    ushort sem_nsems; // 信号量集中信号量的个数
    time_t sem_otime; // 最近一次调用 semop 函数的时间
    time_t sem_ctime; // 最近一次改变该信号量集的时间
};
```

sem 结构记录了一个信号量的信息，其定义如下：

```
struct sem
{
    ushort semval; // 信号量的值
```

```
pid_t smepid;      //最近一次访问资源的进程 ID  
ushort semncnt;   //等待可利用资源出现的进程数  
ushort semzcnt;   //等待全部资源可被独占的进程数  
}
```

(1) 信号量集的创建与打开

在程序使用信号量之前，必须首先创建信号量。

```
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/sem.h>  
  
int semget(key_t key,int nsems,int semflg);
```

函数 **semget** 用于创建一个新的信号量集或打开一个已存在的信号量集。其作用由参数 **key** 和 **flag** 决定，相应约定与 **shmget** 类似。

参数说明：

key: 要创建或要打开的信号量集的键。

nsems: 要创建或者要访问的信号量集中信号量的数目。此参数只在创建一个新的信号量集时有效。

semflg: 指定不同的选项和权限位的标志。可以为 **IPC_CREATE**,**IPC_EXCL**.

该函数调用成功时，返回值为信号量的引用标示符；调用失败时，返回-1，同时 **error** 被设置值。当调用该函数创建一个信号量集时，他相应的 **semid_ds** 结构被初始化。**ipc_perm** 中各个量被设置为相应值，**sem_nsems** 被设置为 **nsems** 所表示的值，**sem_otime** 被设置为 0，**sem_ctime** 被设置为当前时间。

Error=:

EACCESS(没有权限)

EEXIST(信号量集已经存在，无法创建)

EIDRM(信号量集已经删除)

ENOENT(信号量集不存在，同时没有使用 **IPC_CREAT**)

ENOMEM(没有足够的内存创建新的信号量集)

ENOSPC(超出限制)

(2) 对信号量的操作

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(int semid,struct sembuf semoparray[ ],unsigned nsops);
```

参数说明:

(1)semid: 信号量集的引用 ID.

(2) semoparray: 是一个 sembuf 结构数组, sembuf 结构用于指定用于 semop 函数所做操作, 数组 semoparray 元素的个数由参数 nops 指出。

sembuf 结构的定义和设定的操作如下:

```
struct sembuf
{
    ushort sem_num;
    short sem_op;
    short sem_flag;
}
```

sem_num 指要操作的信号量

sem_flag 为操作标记, 于此函数相关的有IPC_NOWAIT和SEM_UNDO。

sem_op用于表示所要执行的操作, 相应取值和含义定义如下:

sem_op>0: 表示进程对资源使用完毕, 交回该资源, 此时信号量集的semid_ds结构的 sem_base.semval将加上sem_op的值。若此时设置了SEM_UNDO位, 则信号量的调整值将减去sem_op的绝对值。

sem_op=0: 表示进程要等待, 直至sem_base.semval变为0.

sem_op<0: 表示进程希望使用资源。此时将比较sem_base.semval和sem_op的绝对值的大小, 如果sem_base.semval大于等于sem_op的绝对值, 说明资源足够分

配给此进程，则 `sem_base.semval` 减去 `sem_op` 的绝对值。若此时设置了 `SEM_UNDO` 位，则信号量的调整值将加上 `sem_op` 的绝对值。如果 `sem_base.semval` 小于 `sem_op` 的绝对值，表示资源不足。若设置了 `IPC_NOWAIT` 位，则函数出错返回，否则 `semid_ds` 结构中的 `sem_base.semncnt` 加 1，进程等待直至 `sem_base.semval` 大于等于 `sem_op` 的绝对值。`sem_base.semval` 大于等于 `sem_op` 的绝对值活该信号量被删除。

`semoparray` 是一个数组，其中每个元素表示一个操作，由于此函数是一个原子操作，一旦执行就将执行数组中的所有操作。

(3) 信号量的控制

对信号量的控制是通过函数 `semctl` 来实现的

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl( int semid, int semnum, int cmd, union semun arg )
```

参数说明：

`semid` 为信号量集的引用标示符

`semnum` 指明某个特定信号量

`cmd` 表示调用该函数希望执行的操作

`arg` 是 `semun` 的联合，该联合定义如下：

```
union semun
{
    int val ;
    struct semid_ds *buf ;
    ushort array ;
};
```

此联合中各个量的使用情况与参数 `cmd` 设置有关，各个量的具体含义将和参数 `cmd` 的各种

取值一起介绍如下：

GETALL: 获得semid所表示的信号量集中信号量的个数，并将该值存在无符号短整数arg.array。

GETNCNT: 获得semid所表示的信号量集中等待给定信号量锁的进程数目，即semid_ds结构中sem.semncnt的值。

GETPID: 获得semid所表示的信号量集中最后一个使用semop函数的进程ID，即semid_ds结构中sem.sempid的值。

GETVAL: 获得semid所表示的信号量集中semnum所指定信号量的值。

GETZCNT: 获得semid所表示的信号量集中的等待信号量成为0的进程数目，即semid_ds结构中sem.semncnt的值。

IPC_RMID: 删除该信号量。

IPC_SET: 按参数arg.buf指向的结构中的值设置该信号量对应的semid_ds结构。只有有效用户ID和信号量的所有者ID或创建者ID相同的用户进程，以及超级用户进程可以执行这一操作。

IPC_STAT: 获得该信号量的semid.ds结构，保存在arg.buf指向的缓冲区。

SETALL: 以arg.array中的值设置semid所表示的信号量集中信号量的个数。

SETVAL: 设置semid所表示的信号量集中semnum所指定的信号量的值。

5、实验代码

```
*****进程间通信*****
*****/
```

//公司名称：飞凌嵌入式技术有限公司
//描述：客户机与服务器进程间用共享内存通信，用信号量达到同步
//gxneicun.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
```

```
#include<sys/shm.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<signal.h>
#if
defined(_GNU_LIBRARY_) && ! defined(_SEM_SEMUN_UNDEFINED)
#else
union semun
{
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *_buf;
};
#endif
#define SHMDATASIZE 1000
#define BUFFERSIZE (SHMDATASIZE - sizeof(int))
#define SN_EMPTY 0
#define SN_FULL 1
int DeleteSemid=0;
```

```
void server(void);
void client(int shmid);
void delete(void);
void sigdelete(int sginum);
void locksem(int semid,int semnum);
void unlocksem(int semid,int semnum);
```

```
void clientwrite(int shmid,int semid,char *buffer);

int main(int argc, char *argv[])//命令本身就是参数

{

    if(argc<2)

        server();

    else

        client(atoi(argv[1]));

    return 0;

}

void server(void)

{

    union semun sunion;

    int semid,shmid;

    void *shmdata;

    char *buffer;

    semid=semget(IPC_PRIVATE,2,SHM_R|SHM_W);

    DeleteSemid=semid;

    /*删除信号量*/

    atexit(&delete);

    signal(SIGINT,&sigdelete);

    /*初始化信号量*/

    sunion.val=1;//信号量可用

    semctl(semid,SN_EMPTY,SETVAL,sunion);

    sunion.val=0;

    semctl(semid,SN_FULL,SETVAL,sunion);

    printf("%d",SETVAL);
```

```
/*分配共享内存段*/
shmid=shmget(IPC_PRIVATE,SHMDATASIZE,IPC_CREAT|SHM_R|SHM_W);

/*映射到内存空间*/
shmdata=shmat(shmid,0,0);

/*结束进程后自动删除共享内存*/
shmctl(shmid,IPC_RMID,NULL);

*(int *)shmdata=semid;//关键，把信号量放到共享内存中，在下面取出来，

shmdata 是共享内存的首址

buffer=shmdata+sizeof(int);//buffer 指向的是 shmdata 的首地址加上 semid 所占
的空间的下一位地址

printf("server is running with id %d\n",shmid);

while(1)
{
    printf("waiting until full...");
    fflush(stdout);//清空缓冲区
    locksem(semid,SN_FULL);//停下来，等待。
    printf("done.\n");
    printf("message received:%s\n",buffer);
    unlocksem(semid,SN_EMPTY);
}

void client(int shmid)
{
    int semid;
```

```
char *buffer;
void *shmdata;
shmdata=shmat(shmid,0,0);
semid=*(int *)shmdata;
buffer=shmdata+sizeof(int);
printf("client operational:shm id is %d\n",shmid);
while(1)
{
    char input[3];
    printf("\n\nmenu\n1.send a message\n");
    printf("2.exit\n");
    fgets(input,sizeof(input),stdin);
    switch(input[0])
    {
        case '1':clientwrite(shmid,semid,buffer);break;
        case '2':exit(0);break;
    }
}
void delete(void)
{
    printf("master exiting:deleting semaphore.\n");
    if(semctl>DeleteSemid,0,IPC_RMID,0)==-1)
        printf("error releasing semaphore.\n");
}
void sigdelete(int signum)
{
    exit(0);
```

```
}

void locksem(int semid,int semnum)

{

    struct sembuf sb;

    sb.sem_num=semnum;

    sb.sem_op=-1;

    sb.sem_flg=SEM_UNDO;

    printf("%d",sb.sem_flg);

    semop(semid,&sb,1);

}

void unlocksem(int semid,int semnum)

{

    struct sembuf sb;

    sb.sem_num=semnum;

    sb.sem_op=1;

    sb.sem_flg=SEM_UNDO;

    semop(semid,&sb,1);

}

void clientwrite(int shmid,int semid,char *buffer)

{

    printf("waiting until empty...");

    fflush(stdout);

    locksem(semid,SN_EMPTY);

    printf("done.\n");

    printf("enter message:");

    fgets(buffer,BUFFERSIZE,stdin);

    unlocksem(semid,SN_FULL);
```

}

6、编译及运行

编译程序 # gcc gxneicun.c -o gx 本实验也可通过交叉编译在开发板上运行

1. 运行服务器程序 # ./gx
2. 在另一个终端界面运行客户端程序./ gx xxx (xxx 为服务器程序打印的 shmid 值)
3. 查看结果

当运行 ./gx 时显示 16server is running with id 131073

waiting until full...

然后运行 ./gx 131073 后会显示 client operational:shm id is 131073

menu
1.send a message
2.exit
如果选择 1，则在客户机敲什么，服务器上就会显示什么，然后又回到菜单界面

menu
1.send a message
2.exit
1
waiting until empty...4096done.
enter message:fhgjhhjhkjk

menu
1.send a message
2.exit

看服务器上的结果：

waiting until full...4096done.
message received:fhgjhhjhkjk

waiting until full...



第十一章 附录

11-1 Windows XP系统中使用虚拟机搭建开发环境

随着 PC 机主频速度的不断进步以及内存价格的持续降低，在一台 PC 上运行一个或多个虚拟机已经变得越来越轻松。几年前，开发或学习 Linux 系统应用需要额外安装一台独立的主机，但目前已经不需要那样做了，我们可以在 Windows 系统上直接运行一个虚拟 Linux 机，就可以很方便的使用 Linux 系统。这样做的好处是显而易见的，可以在 Windows 和 Linux 之间快速切换，也可以在两者之间共享文件，为学习和开发 Linux 系统带来了很大的便利。

VMware Workstation 是一款功能强大的桌面虚拟计算机软件，使得用户可在单一的桌面上同时运行不同的操作系统，是进行开发、测试、部署新的应用程序的最佳解决方案。VMware Workstation 可在一部实体机器上模拟完整的网络环境，以及可便于携带的虚拟机器，其更好的灵活性与先进的技术胜过了市面上其他的虚拟计算机软件。对于企业的 IT 开发人员和系统管理员而言，VMware 在虚拟网路，实时快照，拖曳共享文件夹，支持 PXE 等方面的特点使它成为必不可少的工具。

11-1-1 在Windows XP中安装VMware Workstation

在光盘中提供了一个 VMware Workstation 8.0.2 版本的安装程序，位于基础资料光盘的“实用工具”目录下。

将 VMware7.rar 拷贝到硬盘上解压，然后双击‘VMware-7.0.1.exe’来安装。

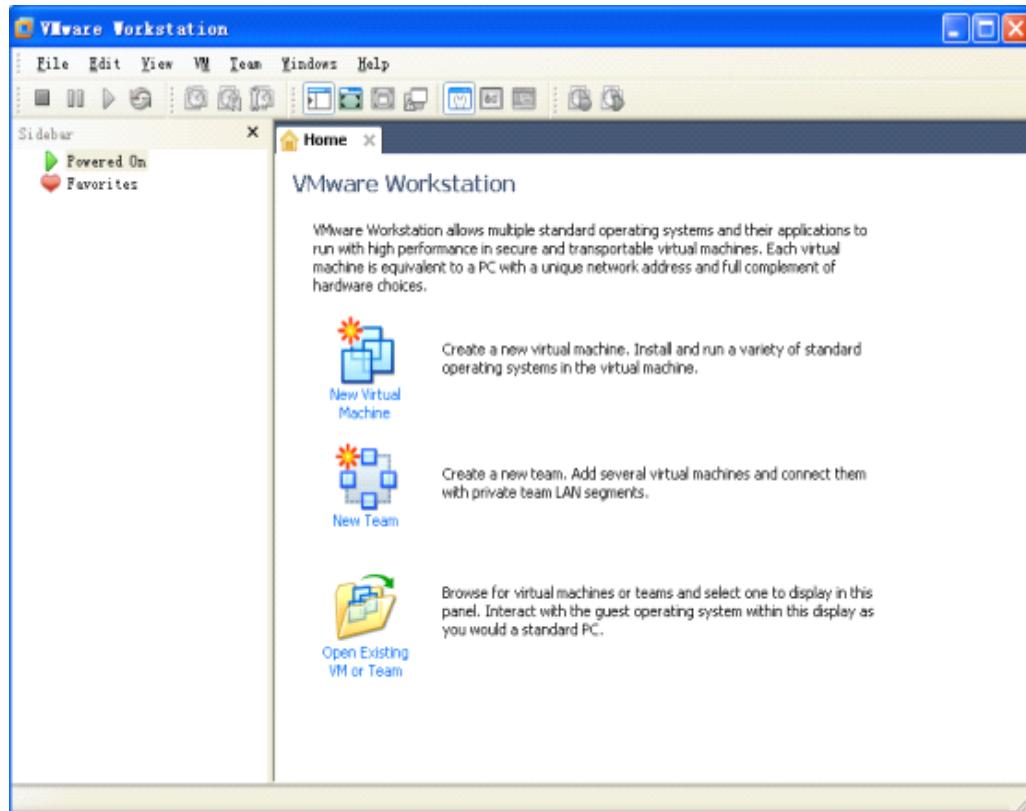
安装过程比较简单，这里将相关步骤从略。

虚拟机安装好以后，就可以在虚拟机中安装相应的系统软件了，比如 Linux、Windows 等等。这样就能在同一台机器上使用多个系统了。

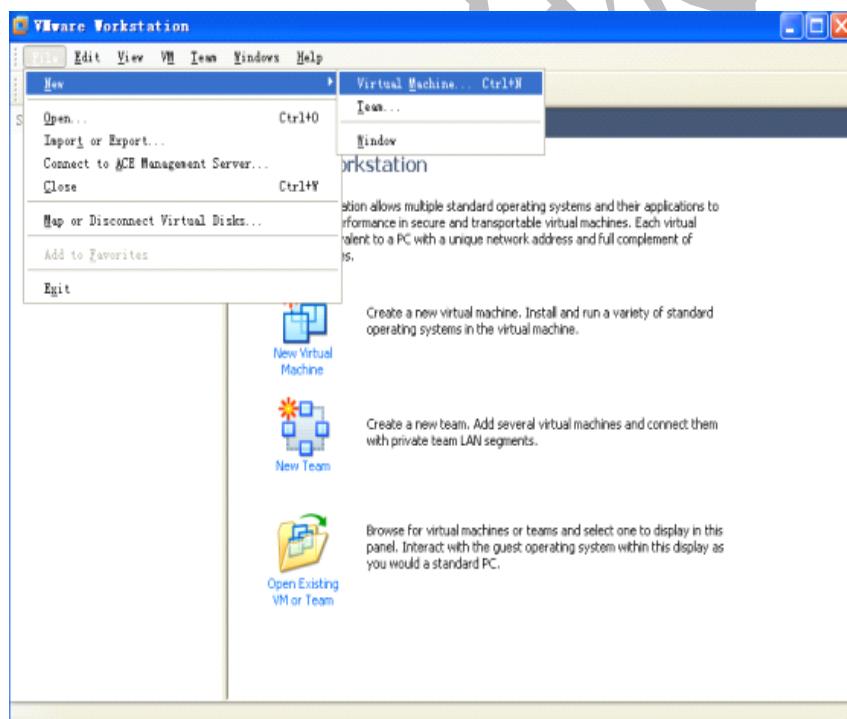
注意：本章的操作环境是 Windows xp，不是 Linux 环境。

11-1-2 VMware新建并设置Ubuntu安装环境

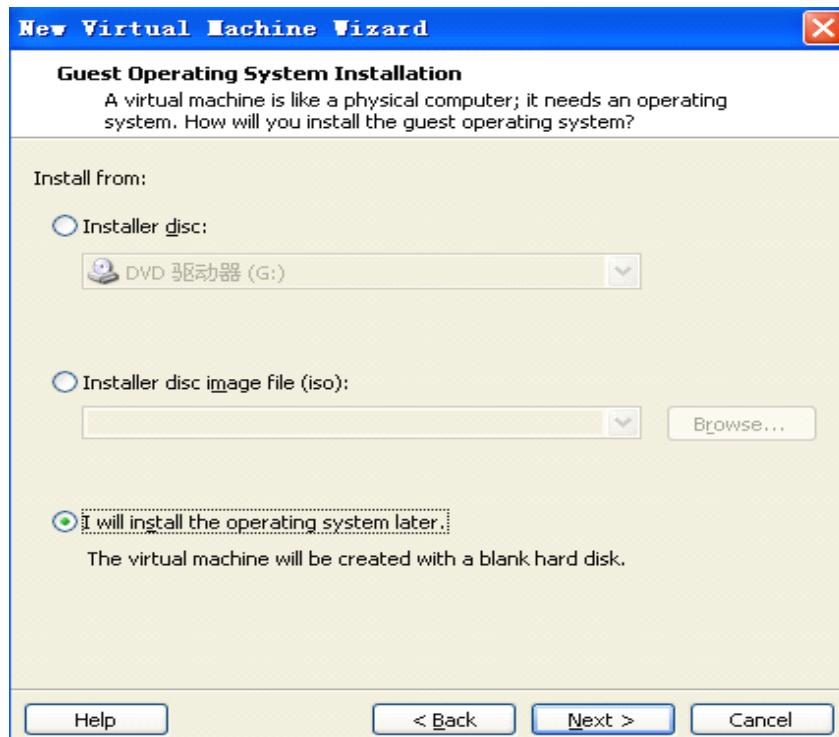
步骤 1. 打开 VMware Workstation。



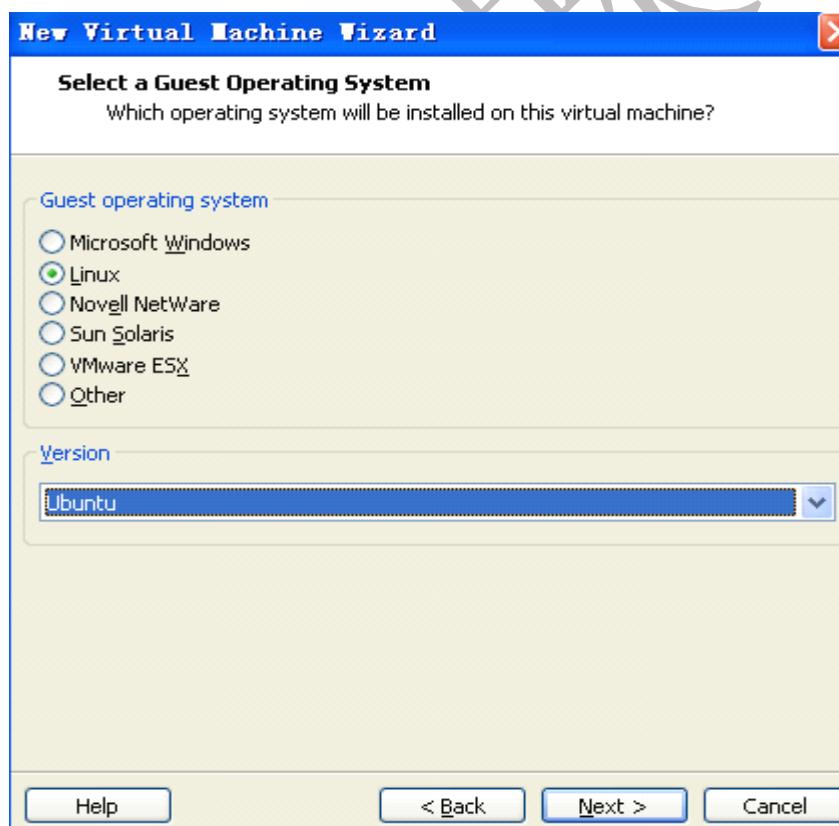
步骤 2. 点击‘File->New->Virtual Machine’新建一个虚拟操作系统。



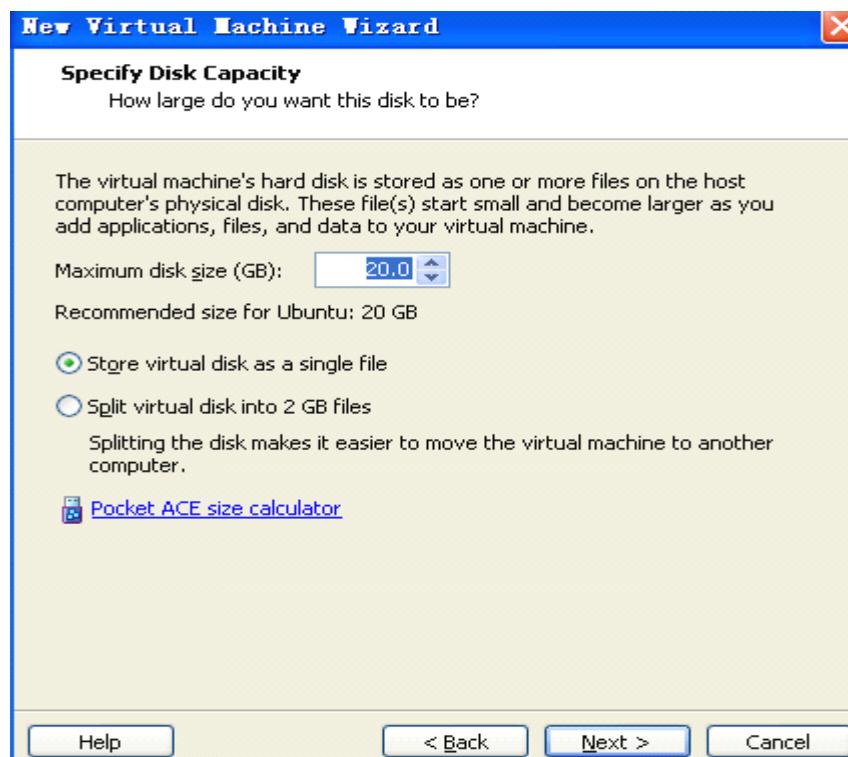
步骤 3. 如下图, 选择 ‘I will install the operating system later.’。



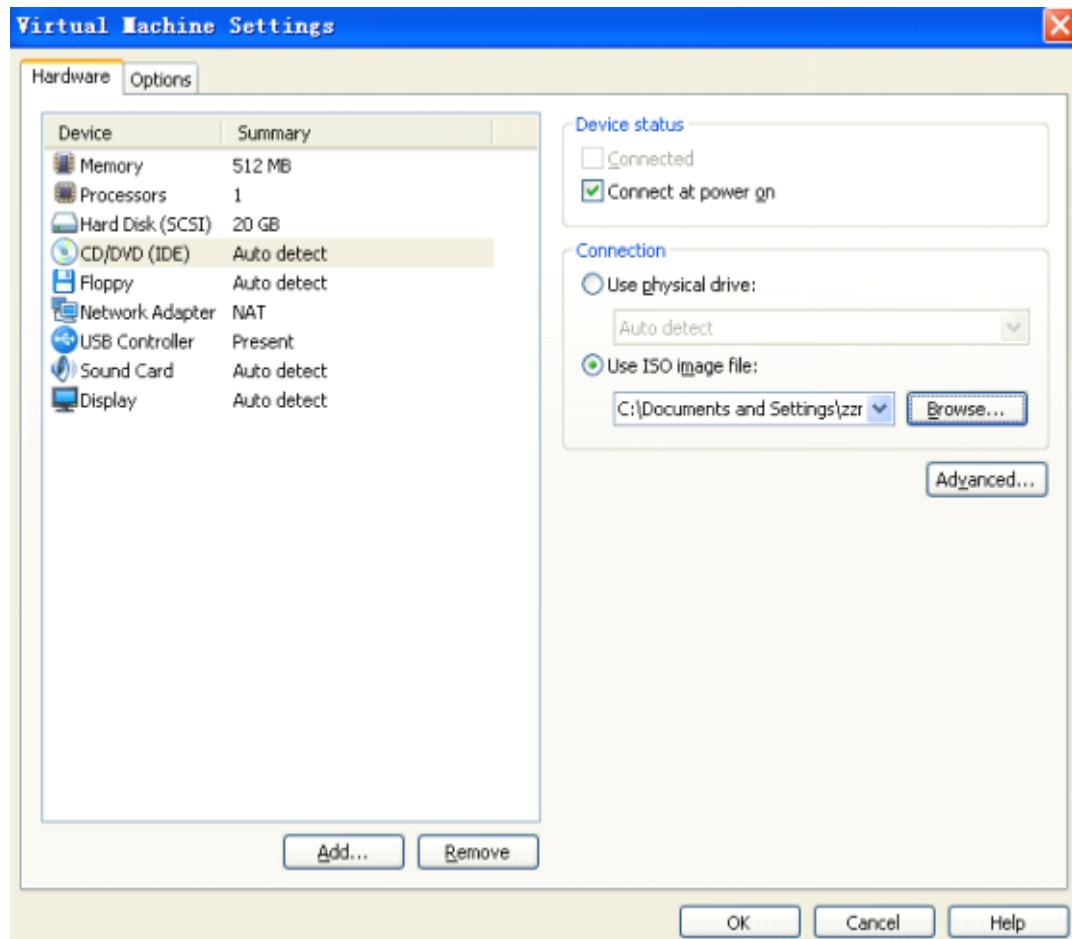
步骤 4. 选择操作系统类型和操作系统版本, 如下图。



步骤 5. 设置虚拟硬盘的选项，如下图。设置好以后，就一路点击“next”即可完成虚拟机的创建。



步骤 6. 虚拟机创建后，设置虚拟机的一些属性，点击 **Edit virtual machine settings**。
步骤 7. 在 **Hardware** 选项签中，设置 **cd/dvd**，选择 **Use ISO image file**, 在 **browse** 中选择 **ubuntu-12.04-desktop-i386.iso**。如下图。选择系统光盘镜像所在位置（一定要拷贝到硬盘上）最后点击“OK”。到这里 VMware 设置的工作就完成了。



接下来，点击虚拟机‘开始按钮’开始安装操作系统。VMware 安装 Ubuntu 的具体过程和硬盘直接安装并无区别，可以将本手册第二章安装方法作为参考。

注意：安装成功后，可能会不能正常启动，如图：

```
M: Skipping non-existing file /cdrom/dists/karmic/restricted/binary-i386/Packages
Removing any system startup links for /etc/init.d/apparmor ...
/etc/rcS.d/S37apparmor
(Reading database ... 120329 files and directories currently installed.)
Removing gdm-guest-session ...
Purging configuration files for gdm-guest-session ...
No value set for '/apps/netbook-launcher/favorites/favorites_list'
Linux ubuntu 2.6.31-14-generic #48-Ubuntu SMP Fri Oct 16 14:04:26 UTC 2009 i686

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

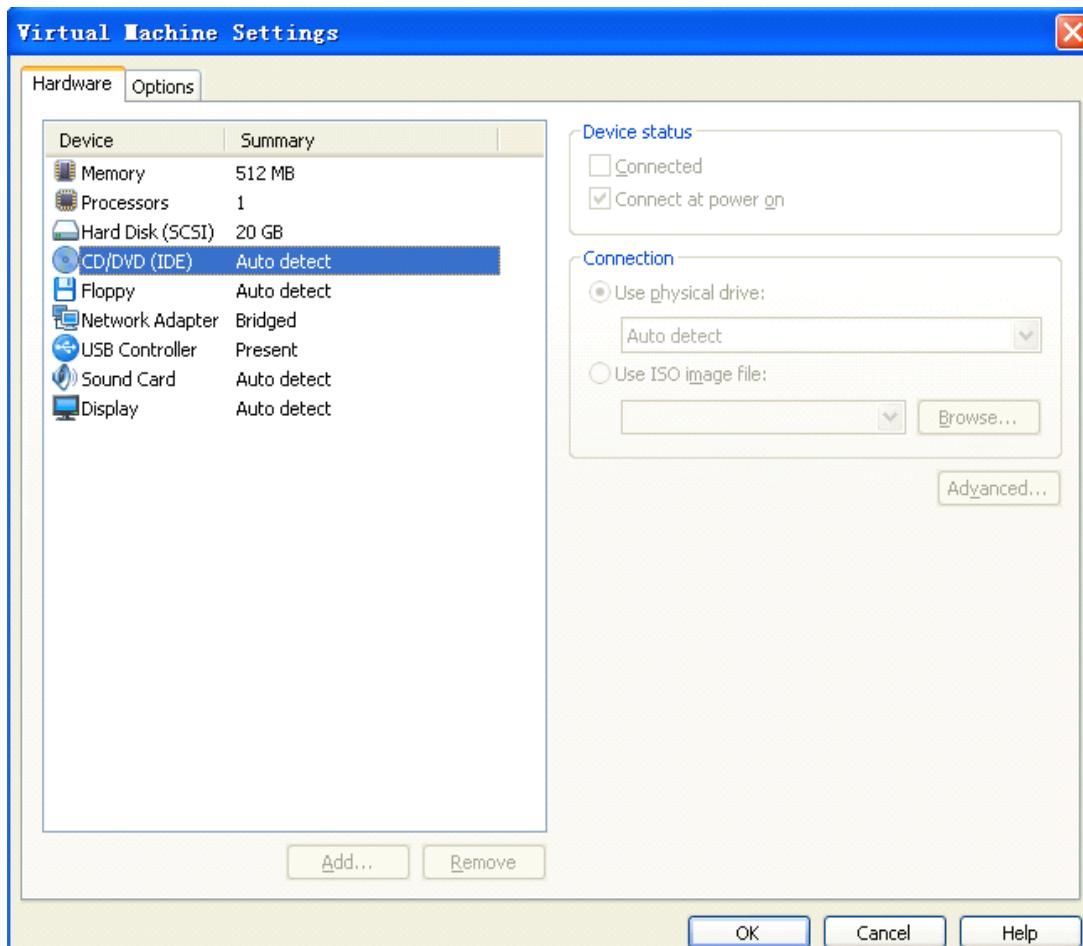
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ubuntu:~$ Broadcast message from root@ubuntu
          (unknown) at 12:13 ...

The system is going down for reboot NOW!
Please remove the disc and close the tray (if any) then press ENTER:
```

点击 Power Off, 使 Ubuntu 关机。

修改虚拟机的 cdrom 设置, 将 cdrom 修改为 auto detect。再次启动 Ubuntu, 就可以正常启动了。设置方法如图:



11-1-3 VMWARE-TOOLS的安装

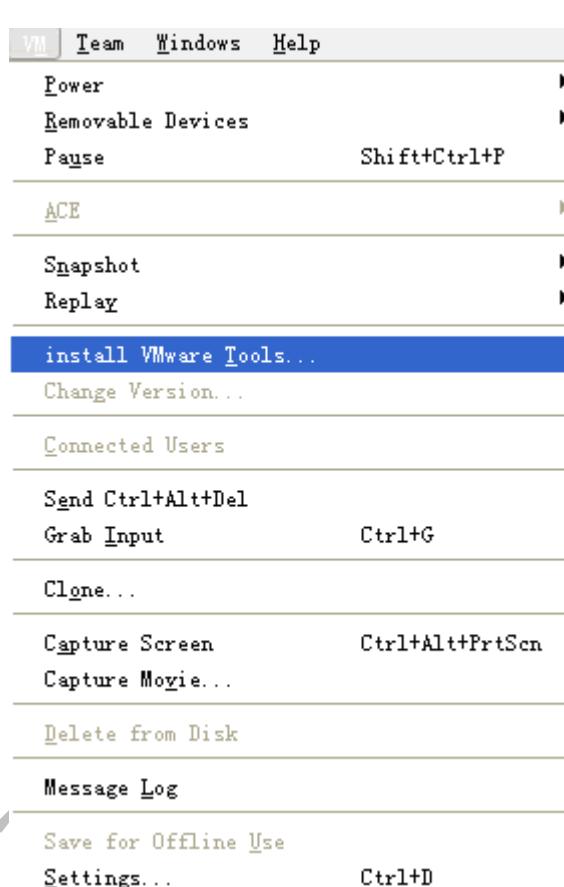
什么是 VMWARE-TOOLS？从字面上看，就是专门为 VMWARE 服务的应用程序。

VMWARE-TOOLS 安装在哪里？安装在 VMWARE 虚拟机安装好的 Linux 上。

VMWARE-TOOLS 能帮咱干嘛？能帮咱们处理很多 WINDOWS 和 Linux 之间、Linux 和 PC 硬件之间兼容性的事情。

这次更新，先给大家演示一下如果安装 VMWARE-TOOLS。

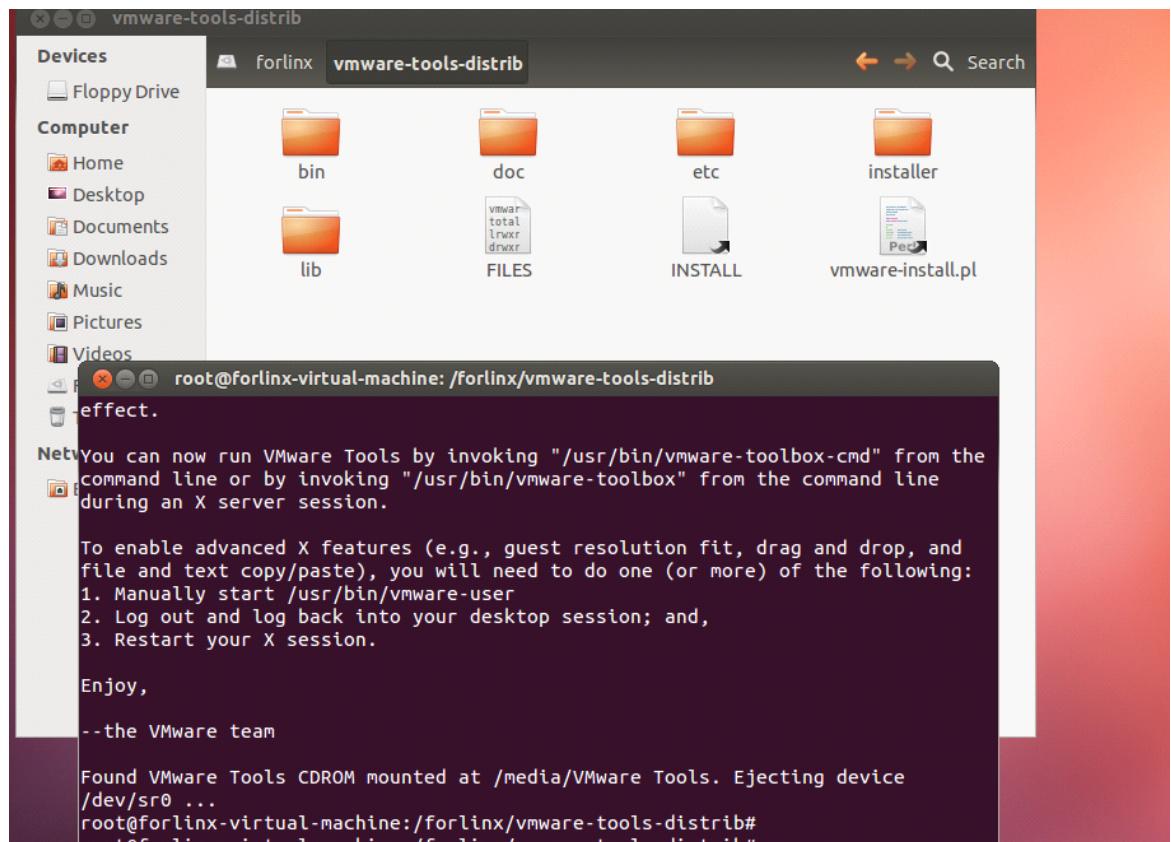
步骤一：点击 VMWARE workstation 标签栏的 VM->install VMware Tools。如图：



接着，在 Ubuntu 中就会看到 VMware Tools 的虚拟光驱。打开光驱，将 VMwareTools-8.1.4-227600.tar.gz 拷贝到 linux 的 /forlinx 目录，然后在 /forlinx 中解压 VMwareTools-8.1.4-227600.tar.gz。使用 Linux 命令行模式，进入刚解压生成的 vmware-tools-distrib 的目录中，执行安装的 shell，命令：`./vmware-install.pl`

安装的过程当中会碰到很多 Y/N 的选择，建议全部选择 Y。

安装按成后，重启 Liunx。



下个版本的手册，添加 VMware-tools 的使用方法和功能。

11-2 使用FTP在XP和Ubuntu间传输文件

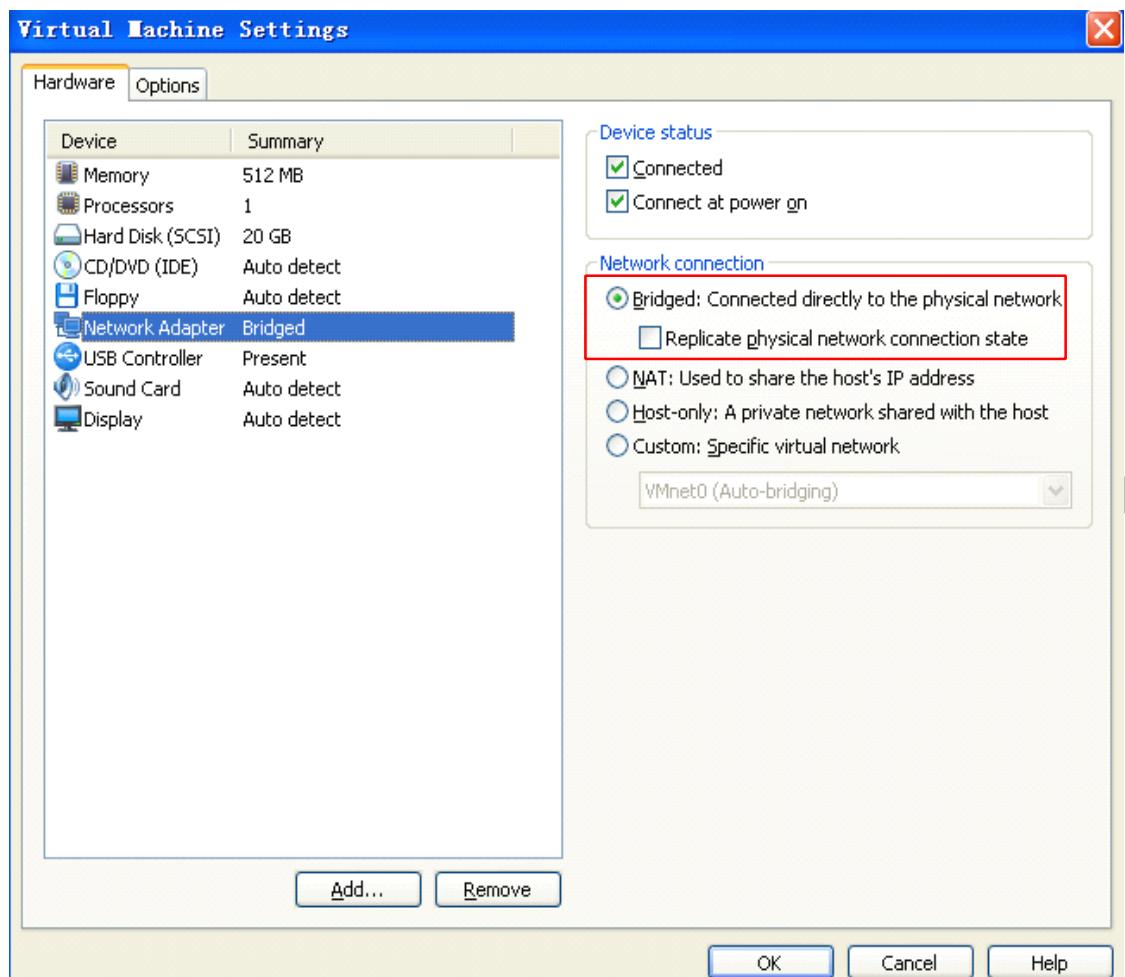
在学习和开发过程中，总会有需要 Linux 和 Windows 之间互相传递文件。方法有很多种，比如 samba 服务器、ftp 服务器、VMware 自带共享功能等等。这里介绍一种通过 ftp 传输文件的方法。

ftp 服务器传输文件，就是通过 FTP 方式在主机（XP）和虚拟机（Ubuntu）之间传输文件，虚拟机 Ubuntu 做服务端，主机 Windows XP 做客户端，下面，将详细介绍使用方法。

11-2-1 设置Ubuntu网络参数

ftp 是基于网络的一种协议，要使用 ftp，那就要保证网络的畅通。所以，让我们先来设置一下 Ubuntu 的网络连接。

步骤 1. 点击 VMware Workstation 菜单 “VM-->Settings...” 会弹出虚拟机设置窗口，选择 Hardware 面板上的 NetworkAdapter，在右边的 Network connection 选项中选择 Bridged 方式。

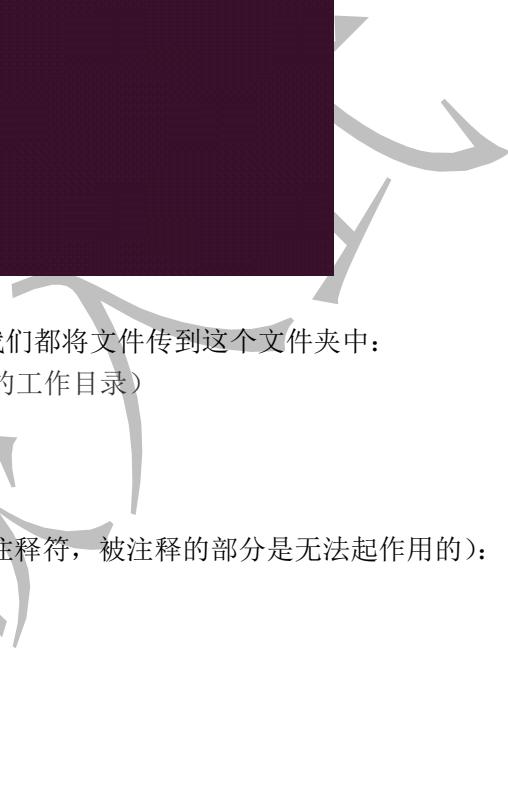


步骤 2. 正确设置 Ubuntu 的网络连接，使之可以连接 internet。

11-2-2 安装Ubuntu的vsftpd服务

- ◆ 在 Ubuntu 中使用 `apt-get install` 命令来升级、安装应用程序，要求 Ubuntu 可以连接到 internet 上。所以，在安装 vsftpd 之前，请确认 Ubuntu 可以访问 internet。
- ◆ 在 Ubuntu 中新建终端，输入下面的命令开始自动下载安装 vsftpd：
`sudo apt-get install vsftpd`

如下图：



```
root@forlinx-virtual-machine: ~
root@forlinx-virtual-machine:~# apt-get install vsftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  vsftpd
0 upgraded, 1 newly installed, 0 to remove and 67 not upgraded.
Need to get 130 kB of archives.
After this operation, 353 kB of additional disk space will be used.
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise/main vsftpd i386 2.3.5-1ubuntu2 [130 kB]
26% [1 vsftpd 33.6 kB/130 kB 26%]
```

- ◆ 在根目录下新建一个文件夹并修改权限，以后我们都将文件传到这个文件夹中：

`mkdir /forlinx` (新建/forlinx 作为我们的工作目录)

`chmod 777 /forlinx`

- ◆ 对 vsftpd 进行配置，输入命令：

`sudo gedit /etc/vsftpd.conf`

主要做以下修改，使以下设置生效：(#符号是注释符，被注释的部分是无法起作用的)：

`anonymous_enable=NO`

`local_enable=YES`

`write_enable=YES`

`local_root=/forlinx`

修改后如图例：

```
# Allow anonymous FTP? (Beware - allowed by default if you comment this out).
anonymous_enable=NO
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
```

`local root=/forlinx`

- ◆ 重新启动 vsftpd 服务：

`sudo /etc/init.d/vsftpd restart`

如果您已经是 root 用户登录系统了，就不需要加 sudo 命令了。

如下图所示：

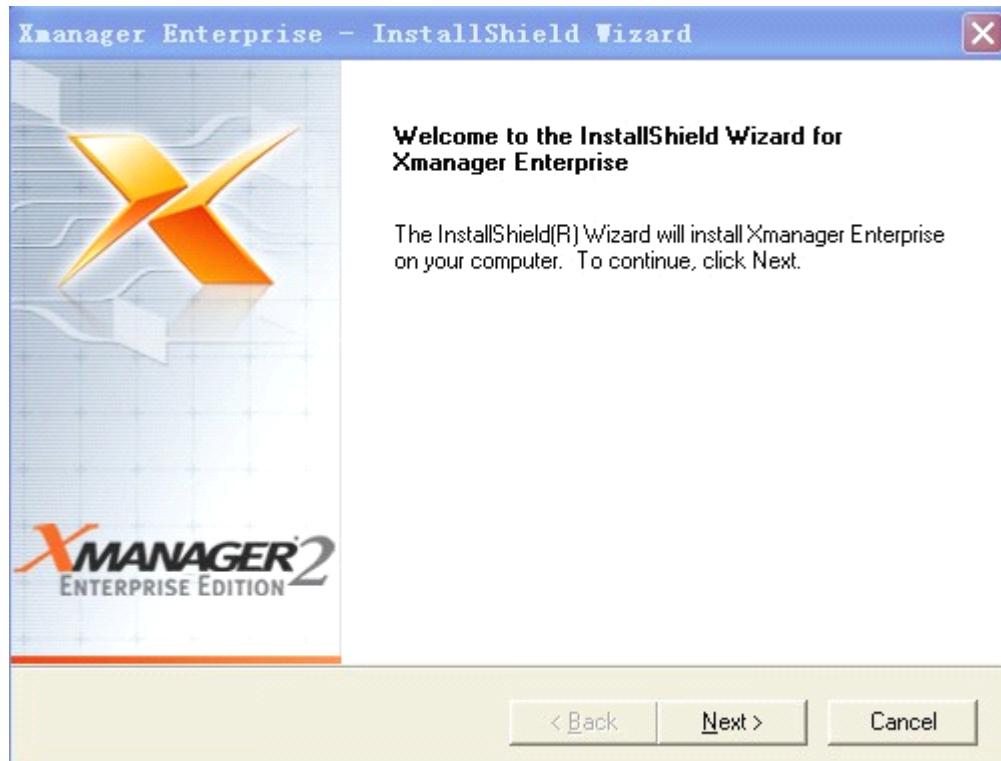
```
root@forlinx-virtual-machine: ~
hwclock-save          udev
irqbalance            udev-fallback-graphics
kerneloops            udev-finish
killprocs              udevmonitor
lightdm                udevtrigger
modemmanager           ufw
module-init-tools      umountfs
networking              umountnfs.sh
network-interface       umountroot
network-interface-container unattended-upgrades
network-interface-security urandom
network-manager         vsftpd
ondemand                whoopsie
plymouth                 x11-common
root@forlinx-virtual-machine:~# /etc/init.d/vsftpd restart
Rather than invoking init scripts through /etc/init.d, use the service(8)
utility, e.g. service vsftpd restart

Since the script you are attempting to invoke has been converted to an
Upstart job, you may also use the stop(8) and then start(8) utilities,
e.g. stop vsftpd ; start vsftpd. The restart(8) utility is also available.
vsftpd stop/waiting
vsftpd start/running, process 3694
root@forlinx-virtual-machine:~#
```

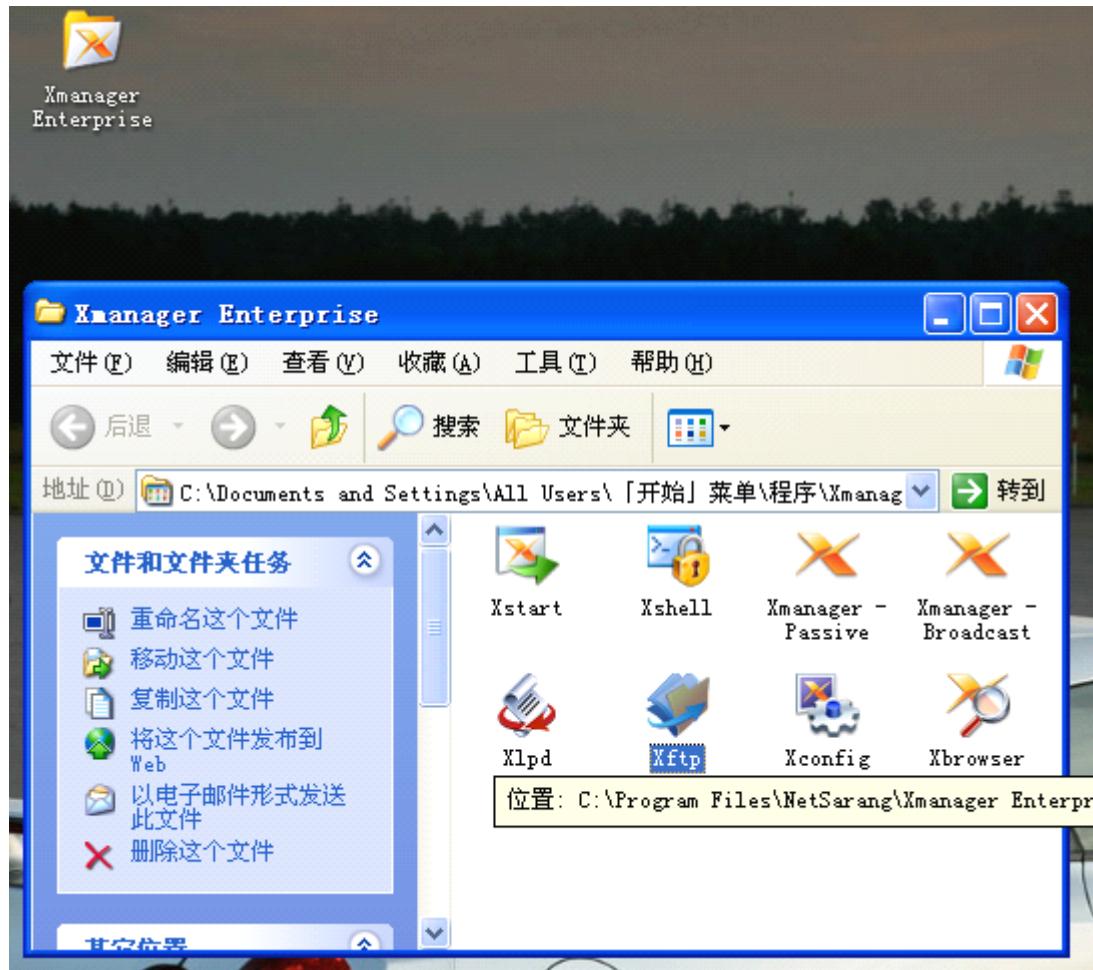
.至此，vsftpd 就安装并启动成功。

11-2-3 安装Windows XP的FTP客户端工具

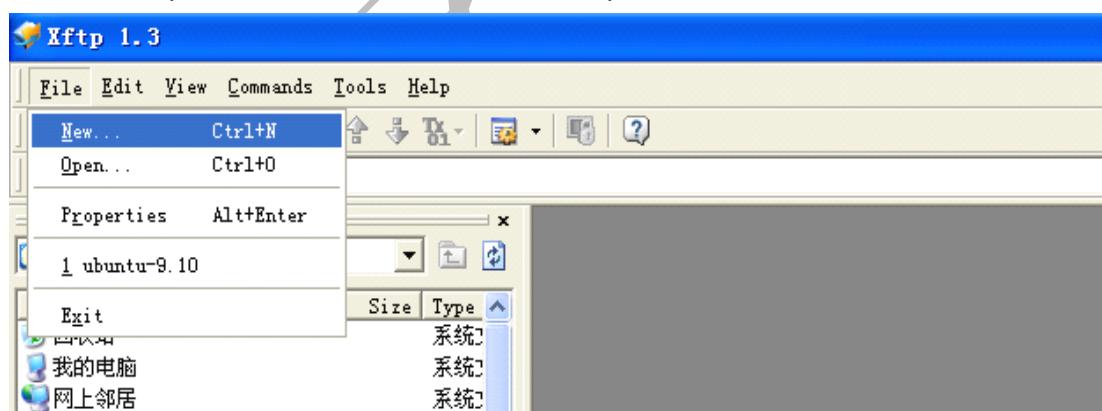
我们在提供了一个 Xmanager Enterprise 软件，其中包含有 ftp 工具；该软件位于基础资料光盘的“实用工具”目录下，解压后双击 xme21.exe 文件开始安装。



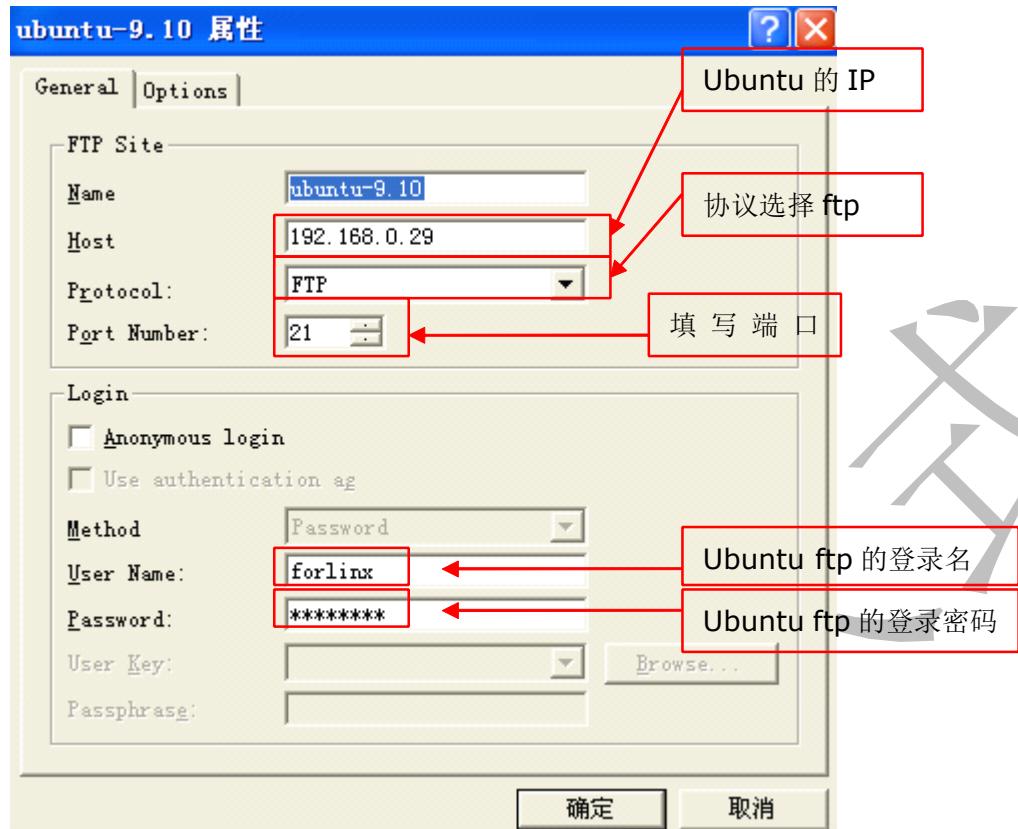
安装完后打开桌面上的 Xmanager Enterprise，里面有 Xmanager 的所有应用程序。在这里我们使用 Xftp。



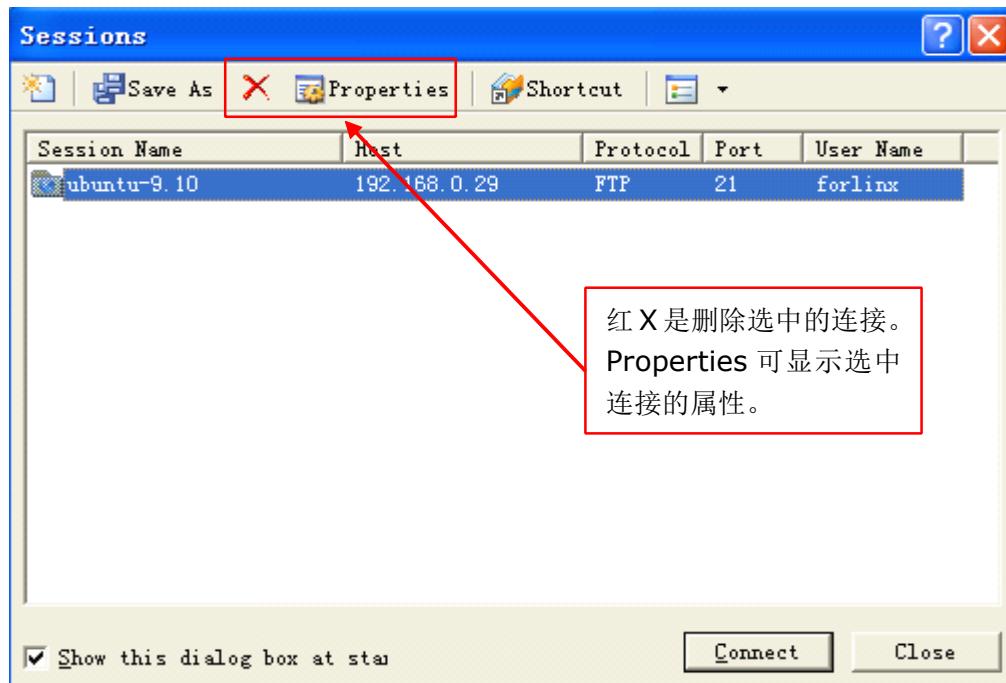
打开 Xftp，点击 File-->New，新建一个 ftp 连接。



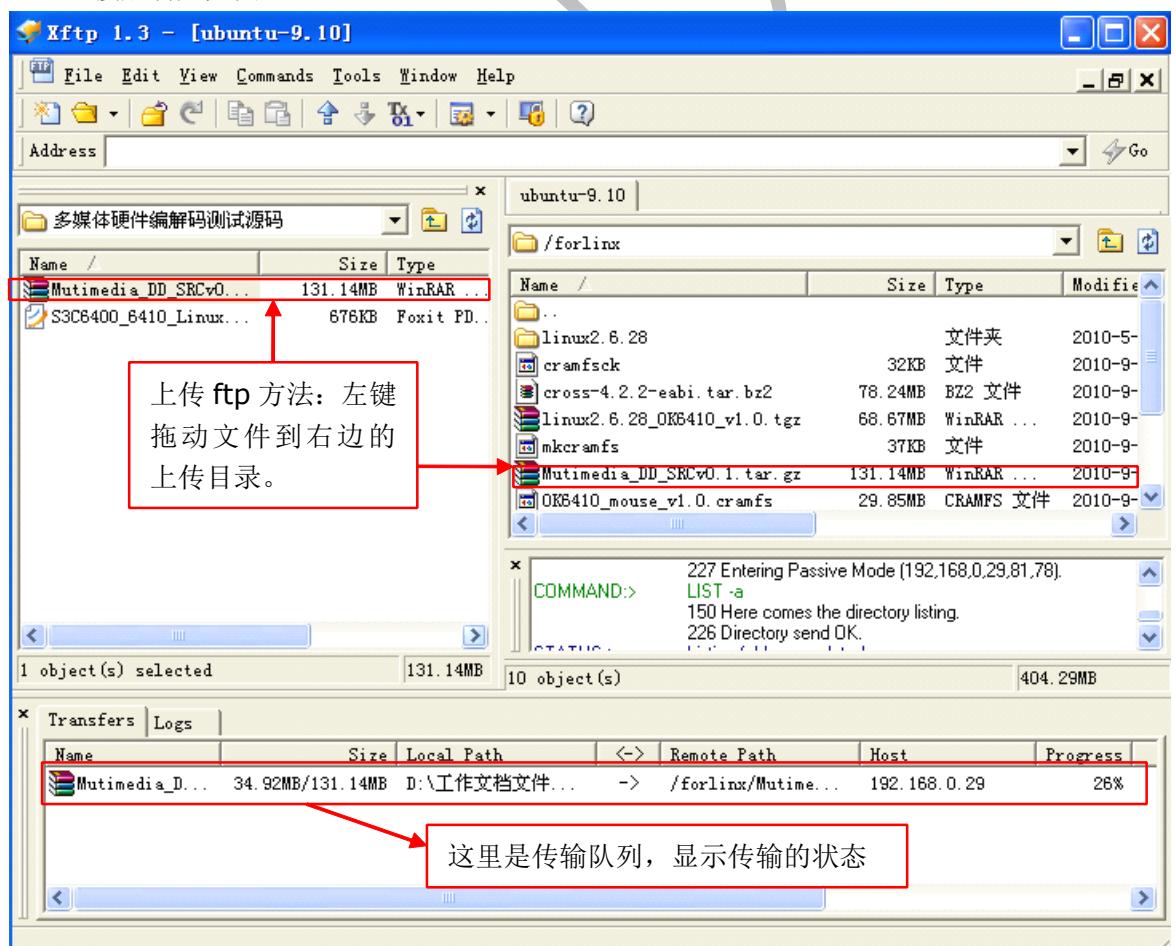
按照如下图所示进行 ftp 连接的属性，然后就可以用该 FTP 软件在主机和虚拟机直接传输文件了。注意这里不能用 root 用户登录。



点击 Connect，连接 Ubuntu 的 ftp 默认目录。



连接成功后如图



文件上传完毕后, 就可以在 /forlinx 目录下看到所上传的文件。如图例:

```
root@forlinx-desktop:/home/forlinx# ls /forlinx/
cramfsck          Mutimedia_DD_SRCv0.1.tar.gz
cross-4.2.2-eabi.tar.bz2  OK6410_mouse_v1.0.cramfs
linux2.6.28        OK6410_touch_v1.0.cramfs
linux2.6.28_OK6410_v1.0.tgz  OK6410_yaffs2_v1.0.tgz
mkcramfs         uboot1.1.6 OK6410 v1.0.tgz
```

11-3 Ubuntu中使用dnw下载

Ubuntu 是 Linux 桌面系统，现在就说说怎么用 Ubuntu 中使用 dnw 下载映像。

dnw 可执行程序由网友提供，在用户基础资料光盘中“\Linux2.6.28\apptest”目录下。

Linux 和 Windows 的 dnw 并不相同，Linux 的 dnw 只有 usb 下载的功能，没有串口的功能。应用程序运行格式为：

```
#dnw path/your_filename
```

11-4 Windows超级终端使用说明

超级终端是一个通用的串行交互软件，很多嵌入式应用的系统有与之交换的相应程序，用这些程序，可以通过超级终端与嵌入式系统交互，使超级终端成为嵌入式系统的“显示器”。

步骤 1. 打开超级中端。开始->程序->附件->通讯->超级终端

步骤 2. 添上名称并确定，在后续的窗口中选择‘COM1’口，确定。

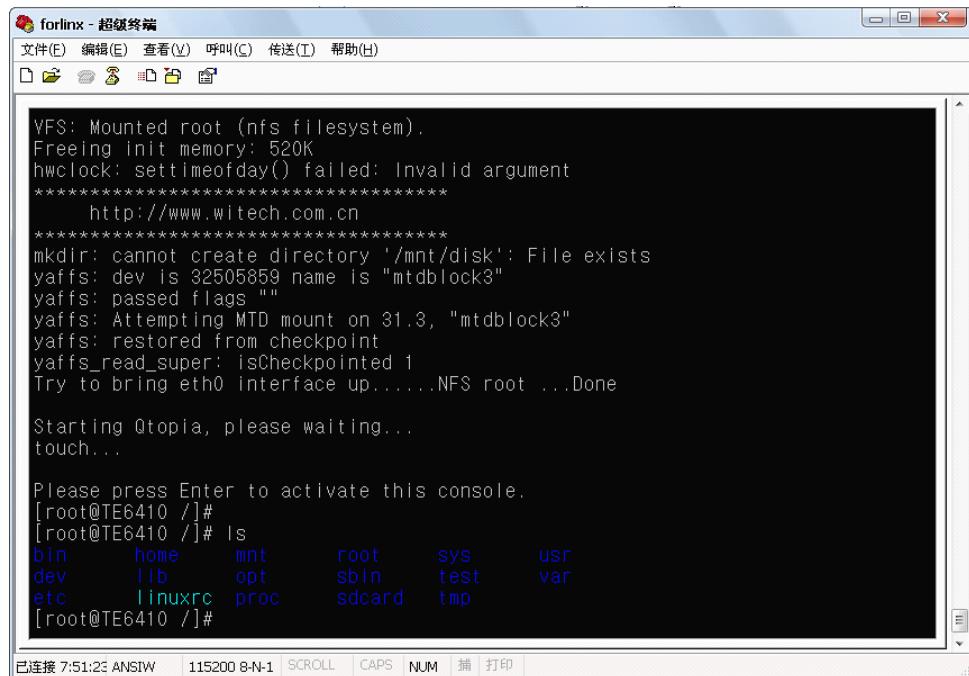




步骤 3. 配置串口属性，按照下图正确进行选择，点击确定。



步骤 4. 连接好 PC 和开发板，并给开发板上电。此时则可在超级终端上显示串口信息。



11-5 字符设备驱动架构分析

相关数据结构：

```
struct cdev {  
    struct kobject kobj;  
    struct module *owner;  
    const struct file_operations *ops;  
    struct list_head list;  
    dev_t dev;  
    unsigned int count;  
};
```

```
struct kobj_map {
    struct probe {
        struct probe *next;
        dev_t dev;
        unsigned long range;
        struct module *owner;
        kobj_probe_t *get;
        int (*lock)(dev_t, void *);
    };
};
```

```
        void *data;
    } *probes[255];
    struct mutex *lock;
};

static struct char_device_struct {
    struct char_device_struct *next;
    unsigned int major;
    unsigned int baseminor;
    int minorct;
    char name[64];
    struct file_operations *fops;
    struct cdev *cdev;           /* will die */
} *chrdevs[CHRDEV_MAJOR_HASH_SIZE];

#define CHRDEV_MAJOR_HASH_SIZE 255
```

下面本文通过以下三个方面以及他们的关联来描述字符设备驱动：

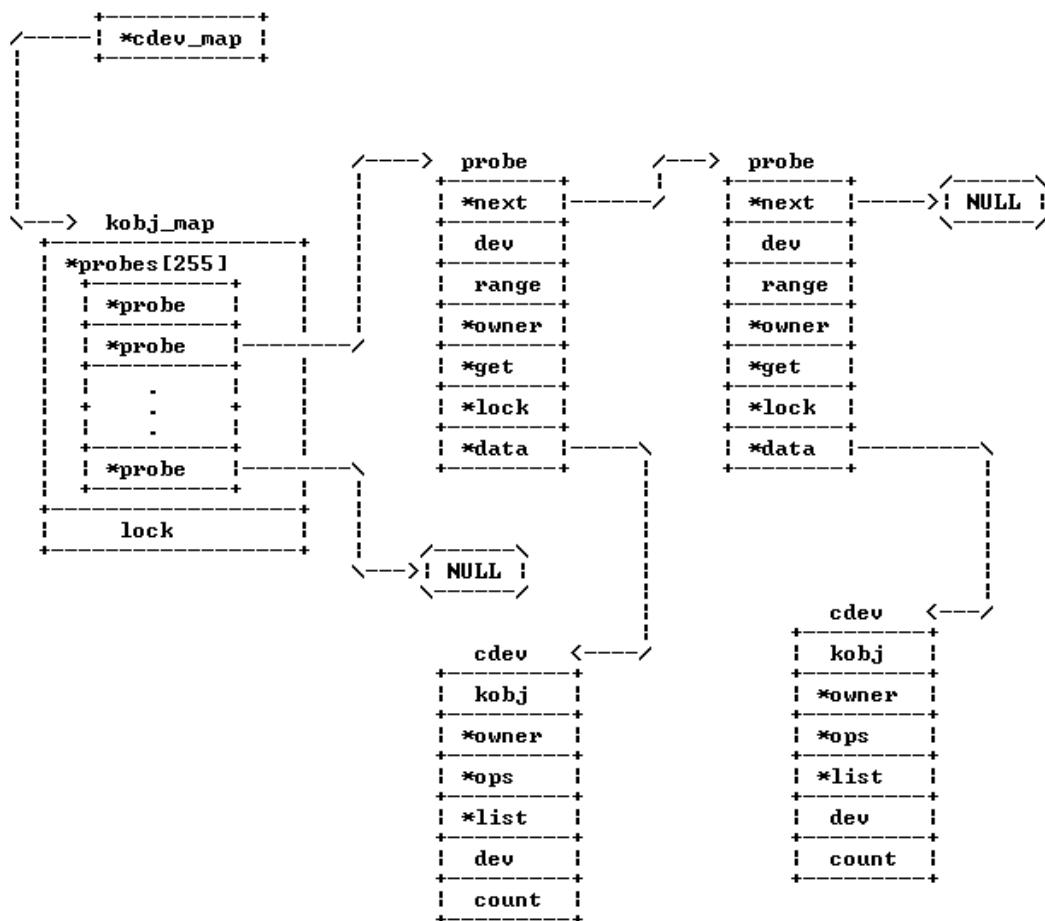
1. 字符驱动模型
2. 字符设备的设备号
3. 文件系统中对字符设备文件的访问

1. 字符驱动模型

每个字符驱动由一个 `cdev` 结构来表示。

在设备驱动模型(device driver model)中，使用 (kobject mapping domain) 来记录字符设备驱动。

这是由 `struct kobj_map` 结构来表示的。它内嵌了 255 个 `struct probe` 指针数组 `kobj_map` 由全局变量 `cdev_map` 引用： `static struct kobj_map *cdev_map;`



相关函数说明：

`cdev_alloc()` 用来创建一个 `cdev` 的对象

`cdev_add()` 用来将 `cdev` 对象添加到驱动模型中,其主要是通过 `kobj_map()`来实现的.

`kobj_map()` 会创建一个 `probe` 对象,然后将其插入 `cdev_map` 中的某一项中,并关联 `probe->data` 指向 `cdev`

`struct kobject *kobj_lookup(struct kobj_map *domain, dev_t dev, int *index)`
根据设备号,在 `cdev_map` 中查找其 `cdev` 对象内嵌的 `kobject`. (`probe->data->kobj`),返回的是 `cdev` 的 `kobject`

2. 字符设备的设备号

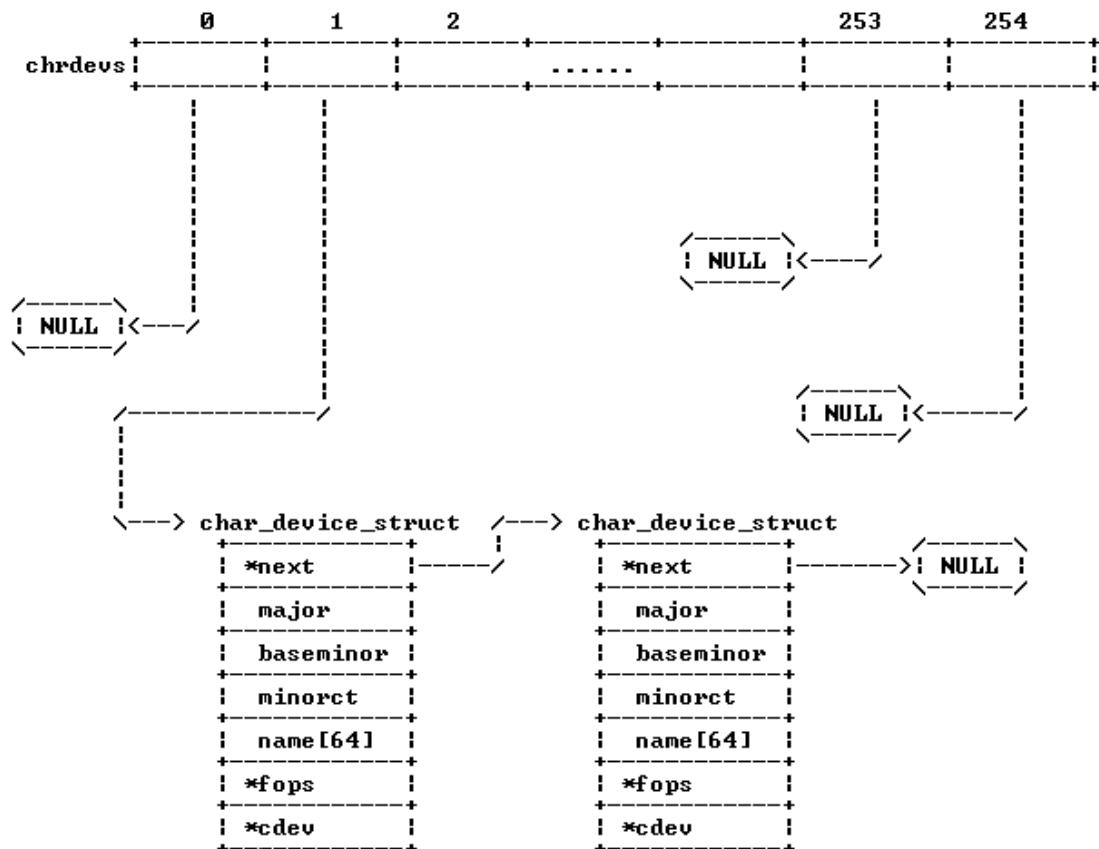
字符设备的主,次设备号的分配：

全局数组 `chrdevs` 包含了 255(`CHRDEV_MAJOR_HASH_SIZE` 的值)个 `struct char_device_struct` 的元素.

每一个对应一个相应的主设备号.

如果分配了一个设备号,就会创建一个 `struct char_device_struct` 的对象,并将其添加到 `chrdevs` 中.

这样,通过 `chrdevs` 数组,我们就可以知道分配了哪些设备号.



相关函数:

`register_chrdev_region()` 分配指定的设备号范围

`alloc_chrdev_region()` 动态分配设备范围

他们都主要是通过调用函数 `register_chrdev_region()` 来实现的

要注意,这两个函数仅仅是注册设备号! 如果要和 `cdev` 关联起来,还要调用 `cdev_add()`

`register_chrdev()` 申请指定的设备号,并且将其注册到字符设备驱动模型中.

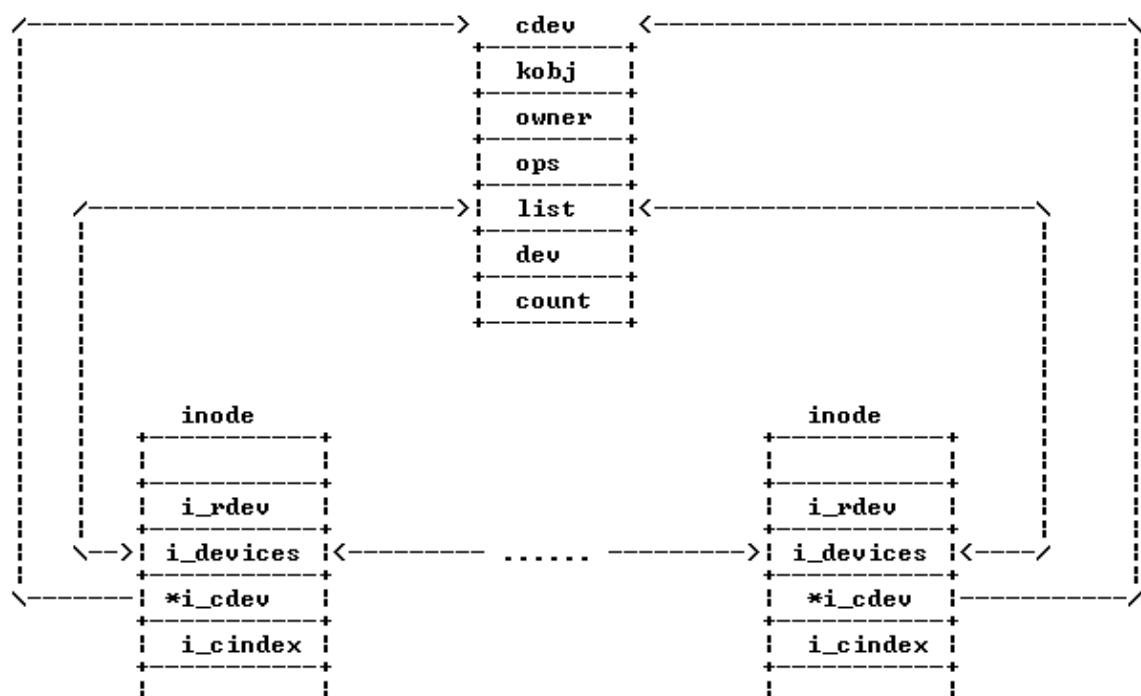
它所做的事情为:

1. 注册设备号, 通过调用 `register_chrdev_region()` 来实现
2. 分配一个 `cdev`, 通过调用 `cdev_alloc()` 来实现
3. 将 `cdev` 添加到驱动模型中, 这一步将设备号和驱动关联了起来. 通过调用 `cdev_add()` 来实现
4. 将第一步中创建的 `struct char_device_struct` 对象的 `cdev` 指向第二步中分配的 `cdev`. 由于 `register_chrdev()` 是老的接口, 这一步在新的接口中并不需要.

3. 文件系统中对字符设备文件的访问

对于一个字符设备文件，其 `inode->i_cdev` 指向字符驱动对象 `cdev`，如果 `i_cdev` 为 `NULL`，则说明该设备文件没有被打开。

由于多个设备可以共用同一个驱动程序。所以，通过字符设备的 `inode` 中的 `i_devices` 和 `cdev` 中的 `list` 组成一个链。



首先，系统调用 `open` 打开一个字符设备的时候，通过一系列调用，最终会执行到 `chrdev_open`。（最终是通过调用到 `def_chr_fops` 中的 `.open`，而 `def_chr_fops.open = chrdev_open`。这一系列的调用过程，本文暂不讨论）

```
int chrdev_open(struct inode * inode, struct file * filp)
```

`chrdev_open()` 所做的事情可以概括如下：

1. 根据设备号(`inode->i_rdev`)，在字符设备驱动模型中查找对应的驱动程序，这通过 `kobj_lookup()` 来实现，`kobj_lookup()` 会返回对应驱动程序 `cdev` 的 `kobject`。
2. 设置 `inode->i_cdev`，指向找到的 `cdev`。
3. 将 `inode` 添加到 `cdev->list` 的链表中。
4. 使用 `cdev` 的 `ops` 设置 `file` 对象的 `f_op`
5. 如果 `ops` 中定义了 `open` 方法，则调用该 `open` 方法

6. 返回.

执行完 `chrdev_open()` 之后, `file` 对象的 `f_op` 指向 `cdev` 的 `ops`, 因而之后对设备进行的 `read`, `write` 等操作, 就会执行 `cdev` 的相应操作.

11-6 Minicom 使用指南

11-6-1 minicom 介绍和设置

`minicom` 是 Linux 下的一个友好的串口通信程序, 类似于 Windows 操作系统上超级终端工具。

总览 SYNOPSIS

```
minicom [-somMlwz8] [-c on|off] [-S script] [-d entry] [-a on|off] [-t term]
[-p pty] [-C capturefile] [configuration]
```

描述 DESCRIPTION

`minicom` 是个通信程序, 有点象共享软件 TELIX, 但其源码可以自由获得, 并能够运行于多数 Unix 系统。它包括以下特性: 自动重拨号的拨号目录, 对串行设备 UUCP 格式的 lock 文件的支持, 独立的脚本

语言解释器, 文件捕获, 多用户单独配置, 等等。

命令行参数 COMMAND - LINE

-s 设置, root 使用此选项在 /etc/minirc.dfl 中编辑系统范围的缺省值。使用此参数后, `minicom` 将不进行初始化, 而是直接进入配置菜单。如果因为你的系统被改变, 或者第一次运行 `minicom` 时, `minicom` 不能启动, 这个参数就会很有用。对于多数系统, 已经内定了比较合适的缺省值。

-o 不进行初始化, `minicom` 将跳过初始化代码。如果未复位(reset)就退出了 `minicom`, 又想重启一次会话(session), 那么用这个选项就比较好(不会再有错误提示: modem is locked -- 注)。但是也有潜在的危险: 由于未对 lock 文件等进行检查, 因此一般用户可能会与 uucp 之类的东东发生冲突..... 也许以后这个参数会被去掉。现在姑且假定, 使用 modem 的用户对自己的行为足够负责。

-m 用 Meta 或 Alt 键重载命令键, 在 1.80 版中这是缺省值, 也可以在 `minicom` 菜单中配置这个选项。不过若一直使用不同的终端, 其中有些没有 Meta 或 Alt 键, 那么方便的做法还是把缺省的命令键设置为 Ctrl - A, 当有了支持 Meta 或 Alt 键的键盘时再使用此选项。Minicom 假定 Meta 键发送 ESC 前缀, 而不是设置字符最高位的那一种(见下)。

-M 跟 “**-m**” 一样, 但是假定 Meta 键设置字符高端的第八位(发送 128+字符代码)。

-z 使用终端状态行。仅当终端支持，并且在其 termcap 或 terminfo 数据库入口中有相关信息时才可用。

-l 逐字翻译高位被置位的字符，使用此标志，minicom 将不再尝试将 IBM 行字符翻译为 ASCII 码，而是将其直接传送。许多 PC - Unix 克隆不经翻译也能正确显示它们(Linux 使用专门的模式：Coherent 和 Sco)。

-a 特性使用，有些终端，特别是 televideo 终端，有个很讨厌的特性处理(串行而非并行)。minicom 缺省使用 ‘-a on’，但若在用这样的终端，你就可以(必须!)加上选项 ‘-a off’。尾字 ‘on’ 或 ‘off’ 需要加上。

-t 终端类型，使用此标志，可以重载环境变量 TERM，这在环境变量 MINICOM 中使用很方便；可以创建一个专门的 termcap 入口以备 minicom 在控制台上使用，它将屏幕初始化为 raw 模式，这样，连同 ‘-f’ 标志一起，就可以不经翻译而显示 IBM 行字符。

-c 颜色使用，有些终端(如 Linux 控制台)支持标准 ANSI 转义序列色彩。由于 termcap 显然没有对于色彩的支持，因而 minicom 硬性内置了这些转义序列的代码。所以此选项缺省为 off。使用 ‘-c on’ 可以打开此项。把这个标志，还有 ‘-m’ 放入 MINICOM 环境变量中是个不错的选择。

-S 脚本，启动时执行给定名字的脚本。到目前为止，还不支持将用户名和口令传送给启动脚本。如果还使用了 ‘-d’ 选项，以在启动时开始拨号，此脚本将在拨号之前运行，拨号项目入口由 ‘-d’ 指明。

-d 启动时拨打拨号目录中的一项，可以用索引号知名，也可以使用入口项的一个子串。所有其它程序初始化过程结束后，拨号将会开始。

-p 要使用的伪终端，它超载配置文件中定义的终端端口，但仅当其为伪 tty 设备。提供的文件名必须采用这样的形式：(/dev/)tty[p-z][0-f]

-C 文件名，启动时打开捕获文件。

-8 不经修改地传送 8 位字符，“连续”意指未对地点/特性进行真正改变，就不插入地点/特性控制序列。此模式用于显示 8 位多字节字符，不是 8 位自否的语言都需要（例如显示芬兰文字就不需要这个）。minicom 启动时，它首先搜索用于命令行参数的 MINICOM 环境变量，这些参数可在命令行上超载。例如：若进行了如下设置：

```
MINICOM=' -m -c on'  
export MINICOM
```

或者其它等效的设置，然后启动 minicom，minicom 会假定终端有 Meat 键或 Alt 键，并且支持彩色。如果从一个不支持彩色的终端登录，并在启动文件(.profile 或等效文件)中设置了 minicom，而且又不想重置环境变量，那么就可以键入 ‘minicom -c off’，来运行这次没有色彩支持的会话。

使用 USE

Minicom 是基于窗口的。要弹出所需功能的窗口，可按下 Ctrl - A (以下使用 C - A 来表示 Ctrl - A)，然后再按各功能键(a - z 或 A - Z)。先按 C - A，再按‘z’，将出现一个帮助窗口，提供了所有命令的简述。配置 minicom(-s 选项，或者 C - A、0)时，可以改变这个转义键，不过现在还是用 Ctrl - A。

以下键在所有菜单中都可用：

UP arrow - up 或 'k'
DOWN arrow - down 或 'j'
LEFT arrow - left 或 'h'
RIGHT arrow - right 或 'l'
CHOOSE Enter
CANCEL ESCape.

屏幕分为两部分：上部 24 行为终端模拟器的屏幕。ANSI 或 VT100 转义序列在此窗口中被解释。若底部还剩有一行，那么状态行就放在这儿；否则，每次按 C - A 时状态行出现。在那些有专门状态行的终端上将会使用这一行，如果 termcap 信息完整且加了 -k 标志的话。

下面按字母顺序列出可用的命令：

C - A 两次按下 C - A 将发送一个 C - A 命令到远程系统。如果把“转义字符”换成了 C - A 以外的什么字符，则对该字符的工作方式也类似。

A 切换“Add Linefeed”为 on/off。若为 on，则每上回车键在屏幕上显示之前，都要加上一个 linefeed。

B 提供一个回卷(scroll back)的缓冲区。可以按 u 上卷，按 d 下卷，按 b 上翻一页，按 f 下翻一页。也可用箭头键和翻页键。可用 s 或 S 键(大小写敏感)在缓冲区中查找文字串，按 N 键查找该串的下一次出现。按 c 进入引用模式，出现文字光标，就可以按 Enter 键指定起始行。然后回卷模式将会结束，带有前缀'>'的内容将被发送。

C 清屏。

D 拨一个号，或转向拨号目录。

E 切换本地回显为 on/off (若你的 minicom 版本支持)。

F 将 break 信号送 modem。

G 运行脚本(Go)。运行一个登录脚本。

H 挂断。

I 切换光标键在普通和应用模式间发送的转义序列的类型(另参下面 关于状态行的注释)。

J 跳至 shell。返回时，整个屏幕将被刷新(redrawn)。

K 清屏，运行 kermit，返回时刷新屏幕。

L 文件捕获开关。打开时，所有到屏幕的输出也将被捕获到文件中。

M 发送 modem 初始化串。若你 online，且 DCD 线设为 on，则 modem 被初始化前将要求你进行确认。

O 配置 minicom。转到配置菜单。

P 通信参数。允许你改变 bps 速率，奇偶校验和位数。

Q 不复位 modem 就退出 minicom。如果改变了 macros，而且未存盘，会提供你一个 save 的机会。

R 接收文件。从各种协议(外部)中进行选择。若 filename 选择窗口和下载目录提示可用，会出现一个要求选择下载目录的窗口。否则将使用 Filenames and Paths 菜单中定义的下载目录。

S 发送文件。选择你在接收命令中使用的协议。如果你未使文件名选择窗口可用(在 File Transfer Protocols 菜单中设置)，你将只能在一个对话框窗口中写文件名。若将其设为可用，将弹出一个窗口，显示你的上传目录中的文件名。可用空格键为文件名加上或取消标记，用光

标键或 j/k 键上下移动光标。被选的文件名将高亮显示。目录名在方括号中显示，两次按下空格键可以在目录树中上下移动。最后，按 Enter 发送文件，或按 ESC 键退出。

- T** 选择终端模拟：ANSI(彩色)或 VT100。此处还可改变退格键，打开或关闭状态行。
- W** 切换 linewrap 为 on/off。
- X** 退出 minicom，复位 modem。如果改变了 macros，而且未存盘，会提供一个 save 的机会。
- Z** 弹出 help 屏幕。

拨号目录 DIALING DIRECTORY

按下 Ctrl - A、D，会进入拨号目录。可以增减、删除或修改各个项目。选择 “dial”，则会拨打标记项目的电话号码，或者当未作任何标记时高亮显示的项目号码。modem 拨号时，可按 ESC 取消；任何其它按键将关闭拨号窗口，但并不取消拨号。拨号目录将保存在 home 目录下的 “.dialdir” 文件中。可用箭头键可以上下卷动，但也可用 PageUp 或 PageDown 键卷动整页。若没有这些键，可用 Ctrl - B(向后)，以及 Ctrl - F(向前)。可用空格键标记多个项目，若 minicom 不能建立一个连接，它将在此列表中循环进行拨号。目录中标记项目的名字前将显示一个 ‘>’ 符号。

“edit” 菜单不言自明，但这里还是简要介绍一下。

- A - Name** 项目名
- B - Number** 电话号码
- C - Dial string #** 指出用于连接的拨号串。在 Modem 和 dialing 菜单中有三种不同的拨号串(前缀和后缀)可以进行设置。
- D - Local echo** 可为 on 或 off (若你的 minicom 版本支持)
- E - Script** 成功建立连接后必须执行的脚本(参 runscript 手册)
- F - Username** 传给 runscript 程序的用户名。在环境串 “\$LOGIN” 中传送。
- G - Password** 传递为 “\$PASS” 的口令。
- H - Terminal Emulation** 使用 ANSI 或 VT100 模拟。
- I - Backspace key sends** 退格键发送的代码(Backspace 或 Delete)。
- J - Linewrap** 可为 on 或 off。
- K - Line settings** 本次连接的 bps 速率，位数和奇偶设置。速率可选当前值，这样就能用当时正在使用的任何速率值。
- L - Conversion table** 可以指定运行 login 脚本前，此拨号项目应答的任何时候要装入的字符转换表。若此域为空，则转换表保持不变。

edit 菜单还显示了最近一次呼叫此项的日期和时间，及呼叫该项的总次数。但并不允许改变这些值。当进行连接时，它们会自动更新。

配置 CONFIGURATION

通常，minicom 从文件“minirc.df1”中获取其缺省值。不过，若给 minicom 一个参数，它将尝试从文件“minirc.configuration”中获取缺省值。因此，为不同端口、不同用户等创建多个配置文件是可能的。最好使用设备名，如：tty1, tty64, sio2 等。如果用户创建了自己的配置文件，那么该文件将以“.minirc.df1”为名出现在 home 目录中。

按 Ctrl - A、0，进入 setup 菜单。人人都可以改变其中的多数设置，但有些仅限于 root。在此，那些特权设置用星号(*)标记。

Filenames and paths 此菜单定义缺省目录。

A - Download directory 下载的文件的存放位置

B - Upload directory 从此处读取上传的文件

C - Script directory 存放 login 脚本的位置

D - Script program 作为脚本解释器的程序。缺省是“runscript”，也可用其它的东西(如：/bin/sh 或“expect”)。Stdin 和 Stdout 连接到 modem，Stderr 连接到屏幕。若用相对路径(即不以'/'开头)，则是相对于 home 目录，除了脚本解释器以外。

E - Kermit program 为 kermit 寻找可执行程序和参数的位置。命令行上可用一些简单的宏：‘%l’ 扩展为拨出设备的完整文件名，‘%b’ 扩展为当前波特率。

F - Logging options Options to configure the logfile writing.

File Transfer Protocols 此处规定的协议将在按下 Ctrl - A、s/r 时显示。行首的“Name”为将要显示在菜单中的名字。“Program”为协议路径，其后的“Name”则确定了程序是否需要参数，如要传送的文件。“U/D”确定了该项要在“upload/download”菜单中出现。“Fullscr”确定要否全屏运行，否则 minicom 将仅在一个窗口中显示其标准输出。“IO - Red”确定 minicom 要否将程序的标准 io 连接到 modem 端口。“Multi”告诉文件名选择窗口协议能否用一个命令发送多上文件。它对于下载协议无效；如果不使用文件名选择窗口，那么上传协议也会忽略它。老版本的 sz 和 rz 非全屏，并且设置了 IO - Red。但是，有些基于 curses 的版本，至少是 rz，不希望其 stdin 和 stdout 被改向，以及全屏运行。所有文件传输协议都以用户的 UID 运行，但并不是总有 UID=root。对于 kermit，命令行上可用‘%l’ 和‘%b’。在此菜单内，还能规定当提示文件要上传时，要否文件选择窗口，以及每次自动下载开始时要否提示下载目录。如果禁止下载目录提示，将使用 file and directory 菜单中规定的下载目录。

11-6-2 Ubuntu12.04 的minicom

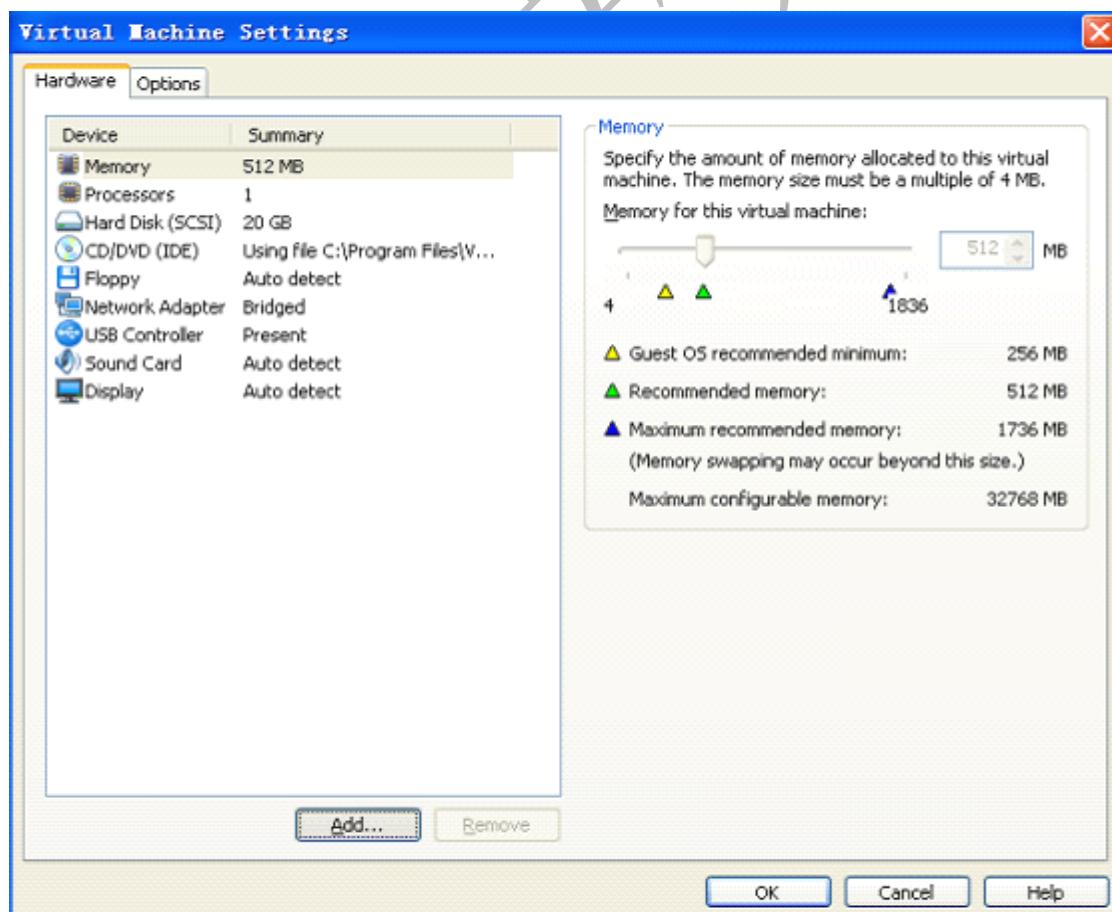
首先，添加和设置虚拟机的硬件接口。

我们要注意一下，在虚拟机中添加、删除或者配置硬件需要把虚拟机的系统关机。

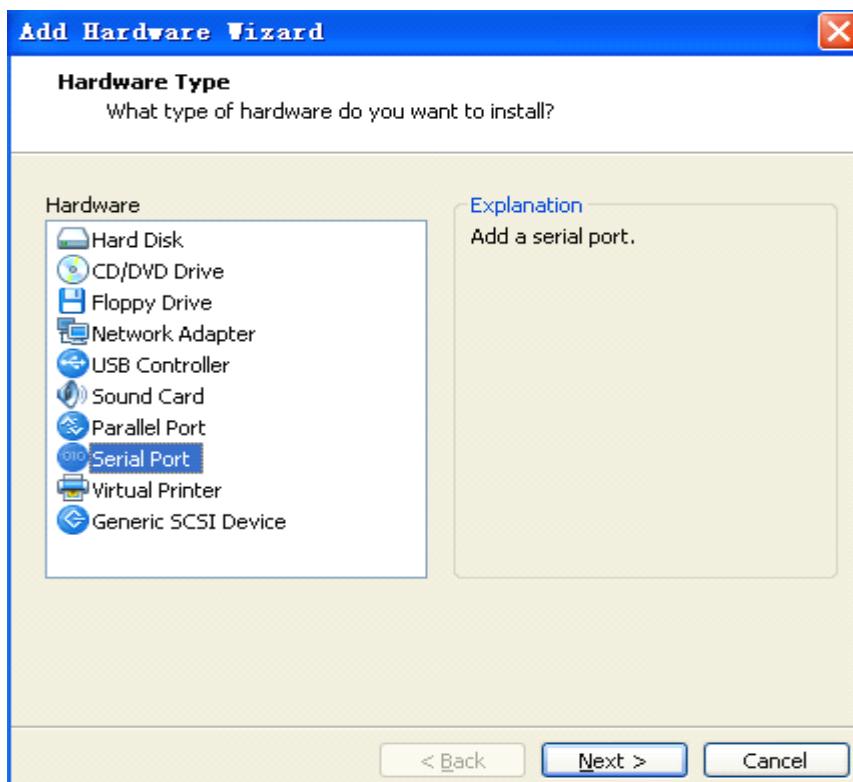
步骤 1. 现在开始操作：虚拟机 vm->Settings



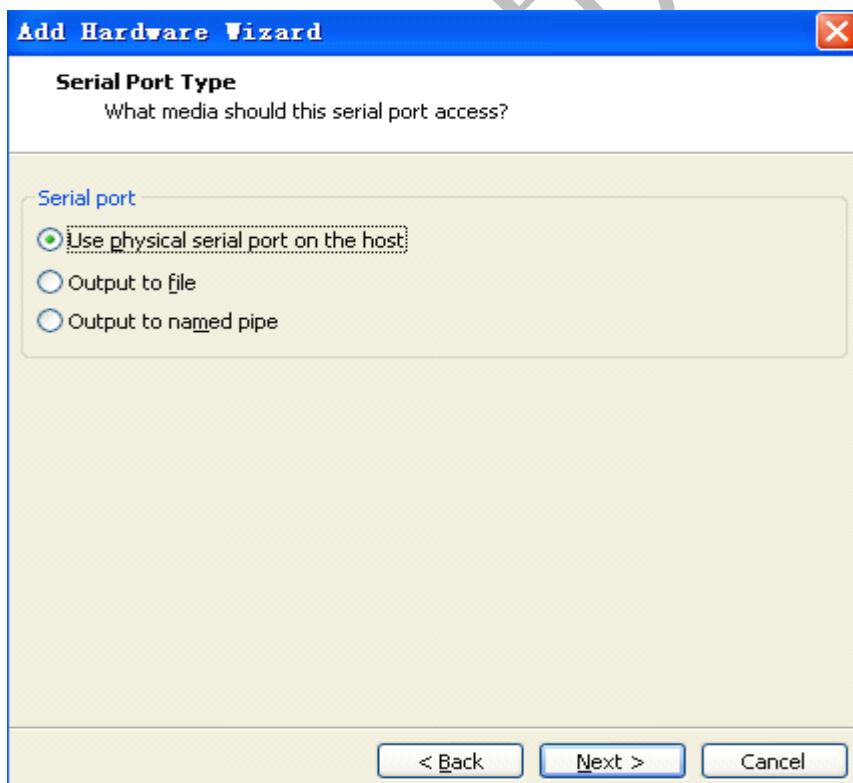
步骤 2. 点击 ‘Add’，准备添加硬件。



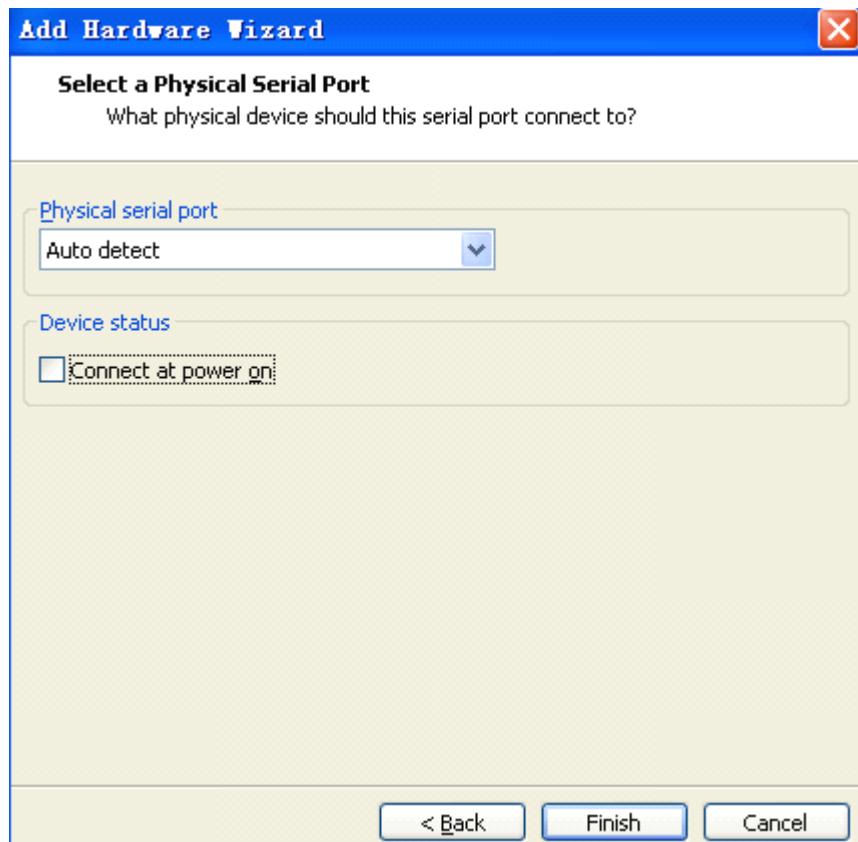
步骤 3. 选择 ‘Serial Port’，点击 ‘next’ 继续。



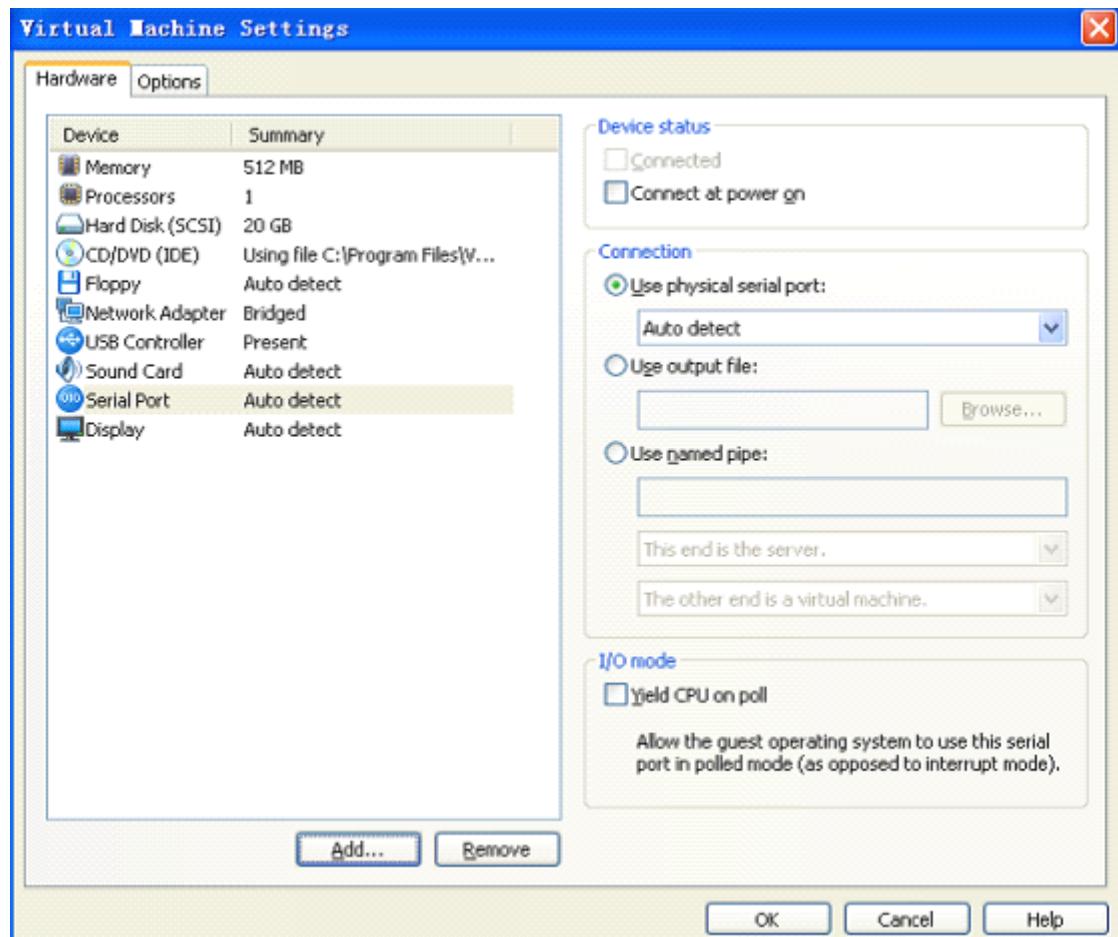
步骤 4. 选择 ‘Use physical serial port on the host’，这样，新添加的虚拟串口就会直接使用 PC 机的物理串口来收发串口信息。点击 ‘next’ 继续。



步骤 5. 选择 ‘Auto detect’，让系统自动选择串口。如果在使用过程当中串口不正确，可以设置为固定的串口号。另外，一个物理串口同一时间内只能由一个软件打开，所以我们不选择 ‘Connect at power on’。后面使用过程当中，我们再介绍如何打开这个连接。



步骤 6. 点击 ‘Finish’，完成添加和设置的工作。接下来，在 vmware 中，会显示出来虚拟串口的属性，如图：



接下来，安装 Ubuntu 中的 minicom。

添加和设置完毕，我们把虚拟机上的 Ubuntu12.04 启动。

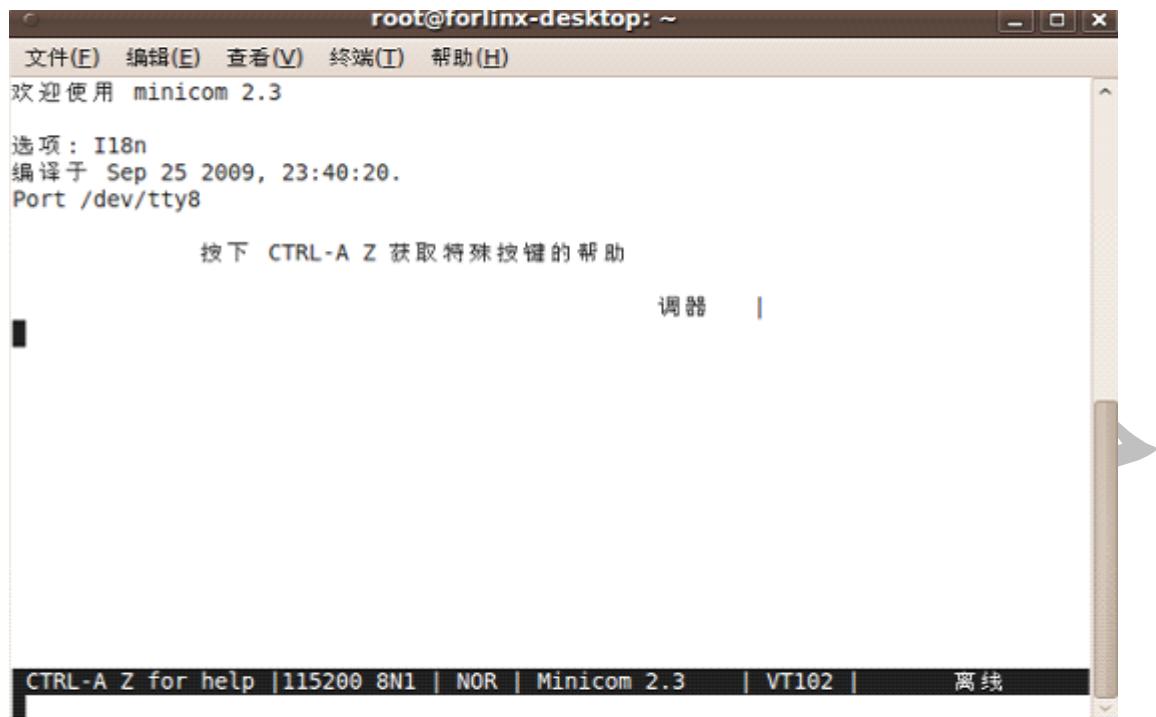
步骤 1. 启动以后，打开终端（[点这里知道如何打开终端](#)）。通过网络安装minicom：

apt-get install minicom

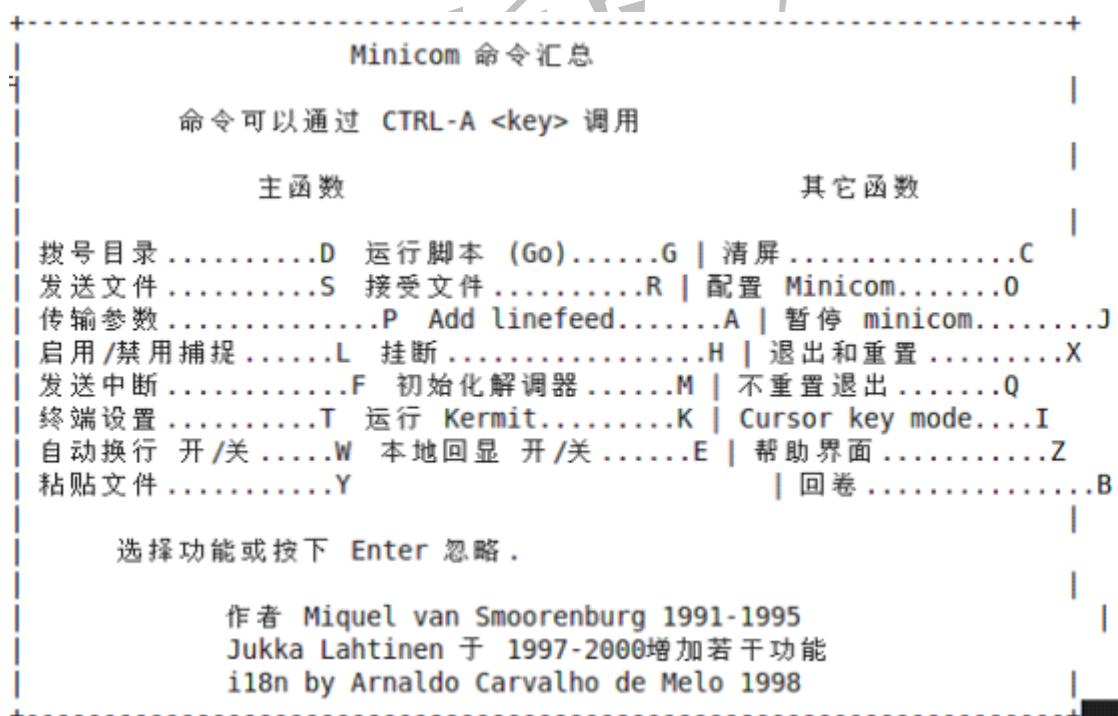
步骤 2. 等待下载安装结束后，在终端中启动 minicom：

minicom

启动后如下图。



步骤 3. 在终端中键入 Ctrl-A，然后键入‘z’，进入 minicom 的功能界面，如同列：



我们开始设置 minicom 的串口。终端里键入 ‘o’，进入设置菜单,如图例:

```
选项 : I18n  
编译于 Sep 25 2009, 23:40:20.  
Port /dev/tty8
```

按下 CTRL-A Z 获得特殊按键的帮助

```
+---[配置]---+  
| 文件名和路径  
| 文件传输协议  
| 串口设置  
| 调制解调器和拨号  
| 屏幕和键盘  
| 设置保存为 dfl  
| 设置保存为 ..  
| 退出  
+-----+
```

步骤 5. 通过键盘的上下左右键, 选择配置菜单的‘串口设置’, 回车, 进入‘串口设置’菜单:

```
+-----+  
| A - 串口设备 : /dev/tty8  
| B - 锁文件的位置 : /var/lock  
| C - 调入程序 :  
| D - 调出程序 :  
| E - Bps/Par/Bits : 115200 8N1  
| F - 硬件数据流控制 : 确定  
| G - 软件数据流控制 : 否  
+-----+  
希望修改哪个设置 ?
```

设置串口设备的路径和硬件数流据控制, 串口设备路径需要根据个人 PC 情况来设置。设置完成后如图例:

```
+-----+  
| A - 串口设备 : /dev/ttyS0  
| B - 锁文件的位置 : /var/lock  
| C - 调入程序 :  
| D - 调出程序 :  
| E - Bps/Par/Bits : 115200 8N1  
| F - 硬件数据流控制 : 否  
| G - 软件数据流控制 : 否  
+-----+  
希望修改哪个设置 ?
```

步骤 6. ‘串口设置’完毕，按 PC 键盘 Esc 建返回上一层配置目录，进入‘调制解调器和拨号’继续设置。

```

-----[调制解调器和拨号参数设置]-----
A - 初始化字符串 ..... ~^M~AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0^M
B - 重置字符串 ..... ^M~ATZ^M~
C - 拨号前缀 #1..... ATDT
D - 拨号后缀 #1..... ^M
E - 拨号前缀 #2..... ATDP
F - 拨号后缀 #2..... ^M
G - 拨号前缀 #3..... ATX1DT
H - 拨号后缀 #3..... ;X4D^M
I - 连接字符串 ..... CONNECT
J - 无连接字符串 .. NO CARRIER           BUSY
      NO DIALTONE          VOICE
K - 停机字符串 ..... ~~+~~ATH^M
L - 拨号取消字符串 ..... ^M

M - 拨号时间 ..... 45
N - 重新拨号前的延迟。 2
O - 尝试次数 ..... 10
P - DTR 丢弃时间 (0=no)。 1
      Q - 自动速率检测 ..... 否
      R - 调制解调器存在 DCD 线路 .. 确
      S - 状态线显示 ... DTE 速度
      T - 多线路 untag .... 否

希望修改哪个设置？ (Return 或 Esc 退出)

```

将 A-初始化字符串 B-重置字符串 K-停机字符串 选项里的字符全部删除。设置完成后如图：

```

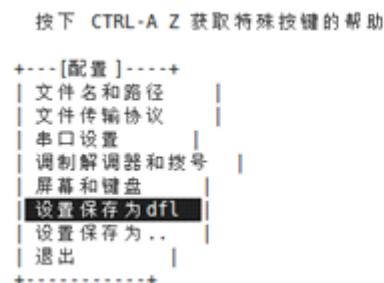
root@forlinx-desktop: /dev
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)
欢迎使用 minicom 2.3
-----[调制解调器和拨号参数设置]-----
A - 初始字符串 .
B - 重置字符串 ...
C - 拨号前缀 #1..... ATDT
D - 拨号后缀 #1..... ^M
E - 拨号前缀 #2..... ATDP
F - 拨号后缀 #2..... ^M
G - 拨号前缀 #3..... ATX1DT
H - 拨号后缀 #3..... ;X4D^M
I - 连接字符串 ..... CONNECT
J - 无连接字符串 .. NO CARRIER           BUSY
      NO DIALTONE          VOICE
K - 停机字符串 ..
L - 拨号取消字符串 ..... ^M

M - 拨号时间 ..... 45
N - 重新拨号前的延迟。 2
O - 尝试次数 ..... 10
P - DTR 丢弃时间 (0=no)。 1
      Q - 自动速率检测 ..... 否
      R - 调制解调器存在 DCD 线路 .. 确
      S - 状态线显示 ... DTE 速度
      T - 多线路 untag .... 否

希望修改哪个设置？ (Return 或 Esc 退出)

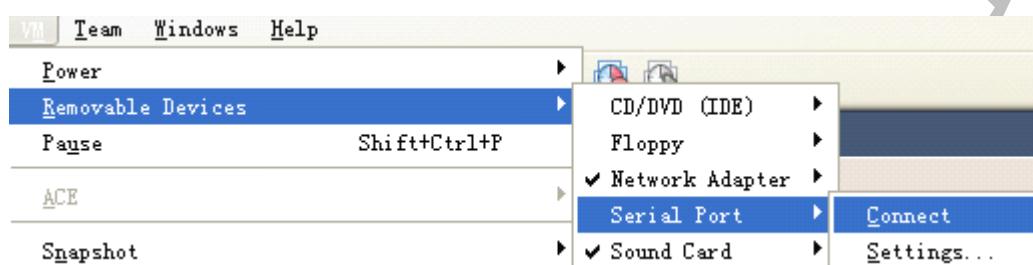
```

步骤 7. 返回到上一层设置目录。将前面设置保存为默认设置，如不保存，设置不会生效。选择‘设置保存为 dfl’。



这样，就完成了 minicom 的设置工作。接下来，用 minicom 连接物理串口。

关闭 Windows 下打开物理串口的软件（比如 dnw、超级终端等），在 vmware 中进行如下操作：VM->Removable Devices->Serial Port->Connect。如图所示：



好了，开始使用 minicom 吧，连接好 PC 和开发板，给开发板上电，这就看到了 U-boot 的启动信息。怎么样，感觉和超级终端没什么区别吧。如果想在 PC 上直接安装的 Ubuntu 上使用 minicom，只要把前面关于 VMware 的设置去掉，就是你要的方法了。

```
U-Boot 1.1.6 (Aug 28 2010 - 11:42:49) for SMDK6410
*****
** u-boot 1.1.6 **
** Updated for TE6410 Board **
** Version 1.0 (10-01-15) **
** OEM: Forlinx Embedded **
** Web: http://www.witech.com.cn **
*****
```

11-7 Linux常用命令详解

Linux 是一个真正的多用户操作系统，它可以同时接受多个用户登录。Linux 还允许一个用户进行多次登录，这是因为 Linux 和 UNIX 一样，提供了虚拟控制台的访问方式，允许用户在同一时间从控制台进行多次登录。虚拟控制台的选择可以通过按下 Alt 键和一个功能键来实现，通常使用 F1 - F6 例如，用户登录后，按一下 Alt - F2 键，用户又可以看到“login:”提示符，说明用户看到了第二个虚拟控制台。然后只需按 Alt - F1 键，就可以回到第一个虚拟控制台。一个新安装的 Linux 系统默认允许用户使用 Alt - F1 到 Alt - F6 键来访问前六个虚拟控制台。虚拟控制台可使用户同时在多个控制台上工作，真正体现 Linux 系统多用户的特性。用户可以在某一虚拟控制台上进行的工作尚未结束时，切换到另一虚拟控制台开始另一项工作。

当然我们也可以在 KDE 环境下使用终端方式输入命令。常见的命令如下：

文件列表 - ls

```
ls #以默认方式显示当前目录文件列表  
ls -a #显示所有文件包括隐藏文件  
ls -l #显示文件属性，包括大小，日期，符号连接，是否可读写及是否可执行  
ls --color=never *.so > obj #不显示文字颜色，将所有 so 文件记录到 obj 文件中
```

目录切换 - cd

```
cd dir #切换到当前目录下的 dir 目录  
cd / #切换到根目录  
cd .. #切换到上一级目录  
cd ../../ #切换到上二级目录  
cd ~ #切换到用户目录，比如是 root 用户，则切换到 /root 下
```

删除 - rm

```
rm file #删除某一个文件  
rm -fr dir #删除当前目录下叫 dir 的整个目录
```

复制 - cp

```
cp source target #将文件 source 复制为 target  
cp /root/source . #将 /root 下的文件 source 复制到当前目录  
cp -av source_dir target_dir #将整个目录复制，两目录完全一样  
cp -fr source_dir target_dir #将整个目录复制，并且是以非链接方式复制，当 source 目录带有符号链接时，两个目录不相同
```

移动 - mv

```
mv source target #将文件 source 更名为 target
```

比较 - diff

diff dir1 dir2 #比较目录 1 与目录 2 的文件列表是否相同，但不比较文件的实际内容，不同则列出

diff file1 file2 #比较文件 1 与文件 2 的内容是否相同，如果是文本格式的文件，则将不相同的内容显示，如果是二进制代码则只表示两个文件是不同的

```
comm file1 file2 #比较文件，显示两个文件不相同的内容
```

回显 - echo

```
echo message #显示一串字符  
echo "message message2" #显示不连续的字符串
```

文件内容查看 - cat

cat file #显示文件的内容，和 DOS 的 type 相同

cat file | more #显示文件的内容并传输到 more 程序实现分页显示，使用命令 less file 可实现相同的功能

more #分页命令，一般通过管道将内容传给它，如 ls | more

设置环境变量 - export

export LC_ALL=zh_CN.GB2312 #将环境变量 LC_ALL 的值设为 zh_CN.GB2312

export DISPLAY=0:0 #通过该设置，当前字符终端下运行的图形程序可直接运行于 Xserver

时间日期 - date

date #显示当前日期时间

date -s 20:30:30 #设置系统时间为 20:30:30

date -s 2002-3-5 #设置系统时期为 2003-3-5

clock -r #对系统 Bios 中读取时间参数

clock -w #将系统时间(如由 date 设置的时间)写入 Bios

容量查看 - du

du #计算当前目录的容量

du -sm /root #计算/root 目录的容量并以 M 为单位

查找 - find

find -name /path file #在/path 目录下查看是否有文件 file

搜索 - grep

grep -ir "chars" #在当前目录的所有文件查找字串 chars，并忽略大小写，-i 为大小写，-r 为下一级目录

编辑 - vi

vi file #编辑文件 file

vi 原基本使用及命令：

输入命令的方式为先按 ctrl+c，然后输入:x(退出)，:x!(退出并保存):w(写入文件)，:w!(不询问方式写入文件)，:r file(读文件 file)，:%s/oldchars/newchars/g(将所有字串 oldchars 换成 newchars)这一类的命令进行操作

读取 - man

man ls #读取关于 ls 命令的帮助

man ls | grep color #读取关于 ls 命令的帮助并通过 grep 程序在其中查找 color 字串

重启 - reboot

reboot #重新启动计算机

关机 - halt

halt #关闭计算机

压缩与解压 - tar

tar xfzv file.tgz #将文件 file.tgz 解压

tar xfzv file.tgz -C target_path #将文件 file.tgz 解压到 target_path 目录下

tar cfzv file.tgz source_path #将文件 source_path 压缩为 file.tgz

tar c directory > directory.tar #将目录 directory 打包成不压缩的 directory.tar

gzip directory.tar #将覆盖原文件生成压缩的 directory.tar.gz

gunzip directory.tar.gz #覆盖原文件解压生成不压缩的 directory.tar。

```
tar xf directory.tar #可将不压缩的文件解包
```

权限设置 - chmod

```
chmod a+x file #将 file 文件设置为可执行, 脚本类文件一定要这样设置一个, 否则得用  
bash file 才能执行
```

```
chmod 666 file #将文件 file 设置为可读写
```

```
chown user /dir #将/dir 目录设置为 user 所有
```

网卡配置 - ifconfig

```
ifconfig eth0 192.168.1.1 netmask 255.255.255.0 #设置网卡 1 的地址 192.168.1.1,  
掩码为 255.255.255.0, 不写 netmask 参数则默认为 255.255.255.0
```

```
ifconfig eth0:1 192.168.1.2 #捆绑网卡 1 的第二个地址为 192.168.1.2
```

```
ifconfig eth0:x 192.168.1.x #捆绑网卡 1 的第二个地址为 192.168.1.x
```

```
ifconfig down eth1 #关闭第二块网卡, 使其停止工作
```

创建设备 - mknod

```
mknod /dev/hda1 b 3 1 #创建块设备 hda1, 主设备号为 3, 从设备号为 1, 即 master 硬盘  
的第一个分区
```

```
mknod /dev/tty1 c 4 1 #创建字符设备 tty1, 主设备号为 4, 从设备号为 1, 即第一个 tty  
终端
```

装载模块 - insmod

```
insmod rt18139.o #装载驱动程序 rt18139.o
```

```
insmod sb.o io=0x280 irq=7 dma=3 dma16=7 mpu_io=330 #装载驱动程序并设置相关的  
irq, dma 参数
```

删除模块 - rmmod

```
rmmod rt18139 #删除名为 rt18139 的驱动模块
```

挂接 - mount

```
mount -t ext2 /dev/hda1 /mnt #把/dev/hda1 装载到 /mnt 目录
```

```
mount -t iso9660 /dev/cdrom /mnt/cdrom #将光驱加载到/mnt/cdrom 目录
```

```
mount -t smb //192.168.1.5/sharedir /mnt -o username=fangtan,password=fangtan  
#将 Windows 的共享目录加载到/mnt/smb 目录, 用户名及密码均为 fangtan
```

```
mount -t nfs 192.168.1.1:/sharedir /mnt #将 nfs 服务的共享目录 sharedir 加载到  
/mnt/nfs 目录
```

卸载 - umount

```
umount /mnt #将/mnt 目录卸载, /mnt 目录必须处于空闲状态
```

```
umount /dev/hda1 #将/dev/hda1 设备卸载, 设备必须处于空闲状态
```

进程查看 - ps

```
ps #显示当前系统进程信息
```

```
ps -ef #显示系统所有进程信息
```

杀死进程 - kill

```
kill -9 500 #将进程编号为 500 的程序杀死
```

11-8 烧写用SD卡和SD读卡器的问答

朋友们会问：开发板能用什么样的 SD 卡烧写系统？

回答：准确来讲，支持 SD 卡、SDHC 卡。

朋友们会问：SD 卡是什么样子？

回答：卡体上标记有“SD”字样的卡。举下图为例：



朋友们会问：SDHC 卡是什么样子？

回答：卡体上标记有“SDHC”字样的卡。举下图为例：



朋友们会问：那其他的卡能用么？

回答：其他的卡通过转接卡，通过正确的烧写，一般是可以启动 SD 卡的。

当然，也有偶尔不能用的。所以强烈建议使用 SD 卡、SDHC 卡。

朋友们会问：操作系统启动以后，其他类型的存储卡支持么？

回答：只要是经过正确的转接，开发板是支持各种存储卡的，不会像烧写那样偶尔挑剔卡的类型。

朋友们会问：对 SD 读卡器有什么要求？

回答：SD 读卡器其实并没有什么特殊的要求，不过为了顺利的进行烧写，还是请多花几块钱买一个有品质保证的 SD 读卡器吧。

朋友们会问：如果没有 SD 读卡器，笔记本上的多合一读卡器可以替代么？

回答：多合一读卡器经常出问题，还是用 SD 读卡器吧。

朋友们会问：飞凌开发板标配中怎么没有 SD 卡和读卡器？

回答：对，确实没有。

[点这里返回一键烧写linux的方法](#)

[点这里返回Linux USB烧写的方法](#)

11-09 Dnw软件的使用简便教程

Dnw 在飞凌基础光盘的路径是”基础光盘\实用工具\dnw.exe”。

光盘中提供的 Dnw 软件版本是 0.6C，这个软件是由三星公司专门为 S3C6410 开发的串口、USB 工具。主要用于对 6410 开发板的串口通信以及 USB 方式烧写。这个版本软件在 XP 下可以稳定运行，WIN7 下暂无测试结果。

在 XP 中打开 Dnw 界面如下：



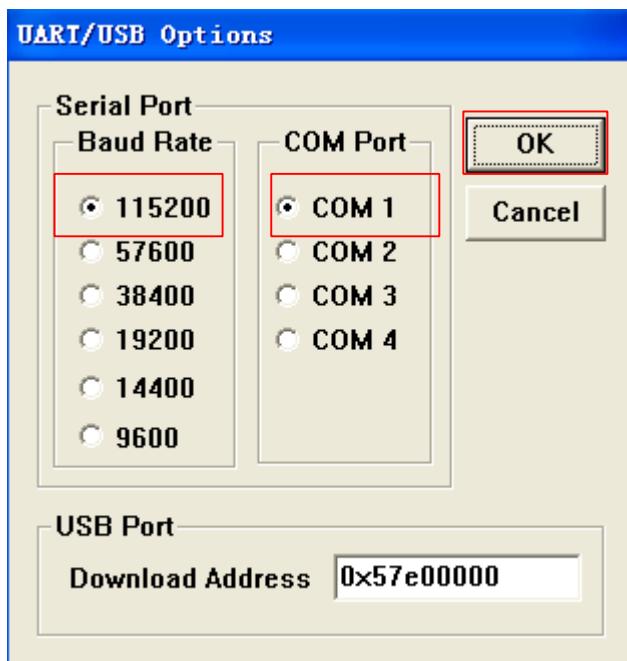
首先，介绍一下串口设置方法：

打开XP的设备管理器，查看本机串口的串口号，如图：



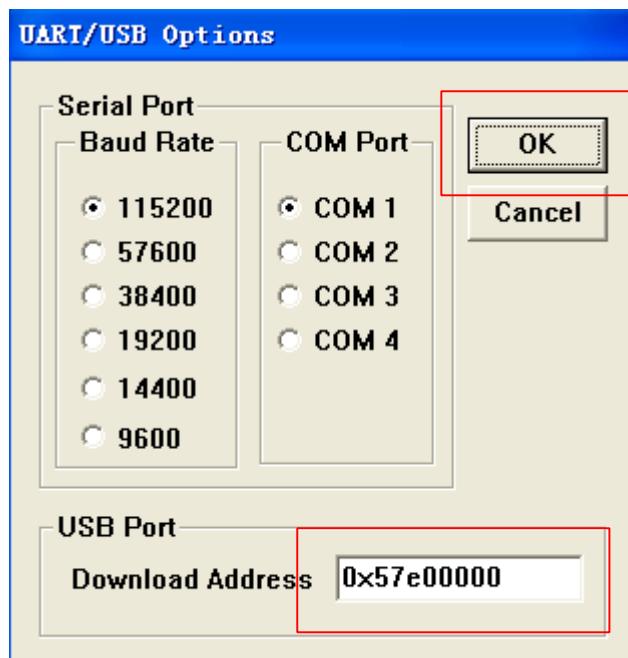
确定了串口号以后，点击“Configuration”->“Option”后，会打开 DNW 的串口、USB 设置

打开设置界面如图：



1. 将波特率调整为 115200
2. 将串口号设置为 PC 串口设备号（如果不清楚自己 PC 的串口号，可以通过 XP 的设备管理器查看）
3. 点击“OK”
4. 在 DNW 主界面选择 Serial port->Connect
如果 DNW 正确的打开了 PC 机的串口（注意这里只和 PC 有关），
会显示如图： [COM1, 115200bps]
- 如果 DNW 没有正确选择打开 PC 机的串口，需要重新设置。

然后，介绍一下 USB 下载地址的设置方法：



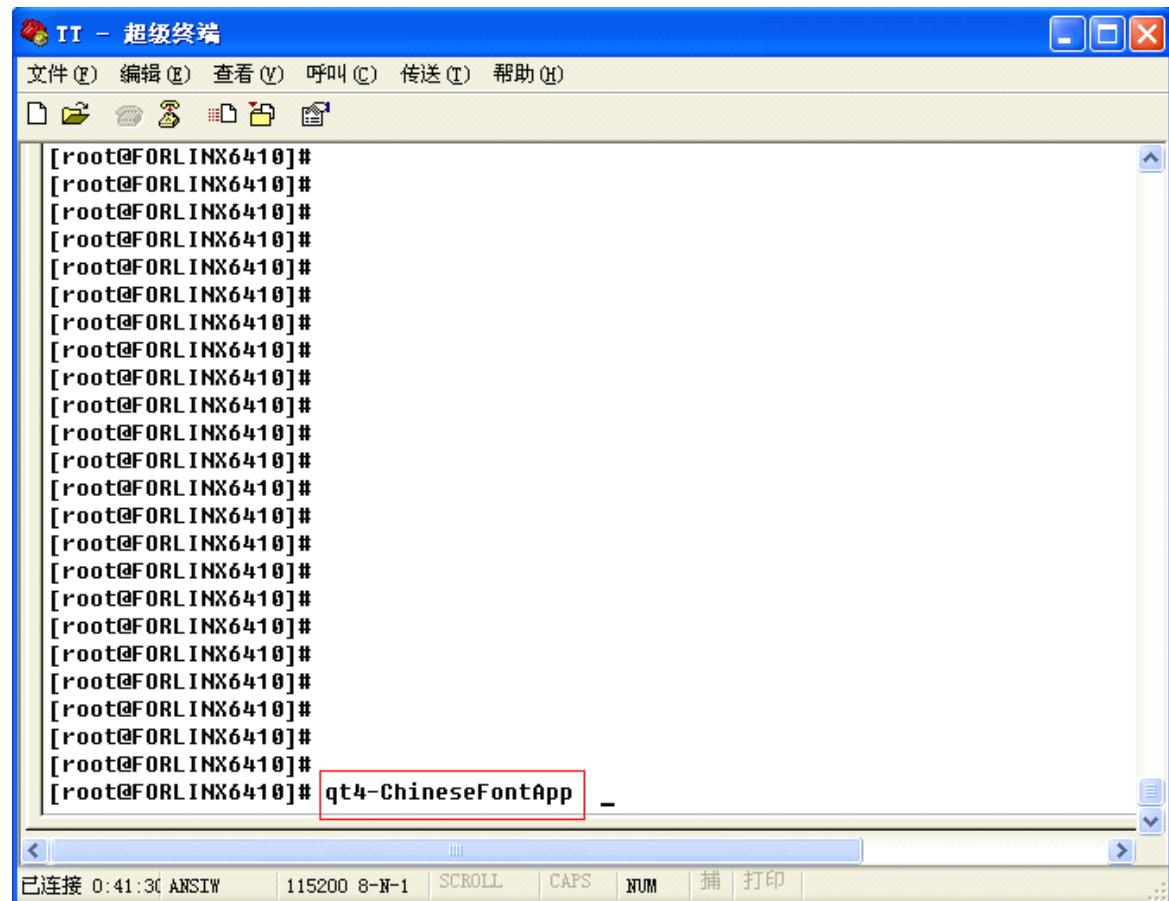
1. 将 USB 下载地址修改为需要的地址
2. 点击“OK”。

11-10 Qt4.7 支持中文显示

Yaffs2 文件系统 Qt4.7 库部分增加了中文字体支持，用户可以使用文泉驿字体显示中文字体，目前我们支持 12, 13, 15, 16 号，粗体和细体字体，字体文件位于 /opt/qt-4.7.1/lib/fonts/ 下面，您可以在 DNW 或者超级终端里面执行 # qt4-ChineseFontApp 命令测试中文显示：

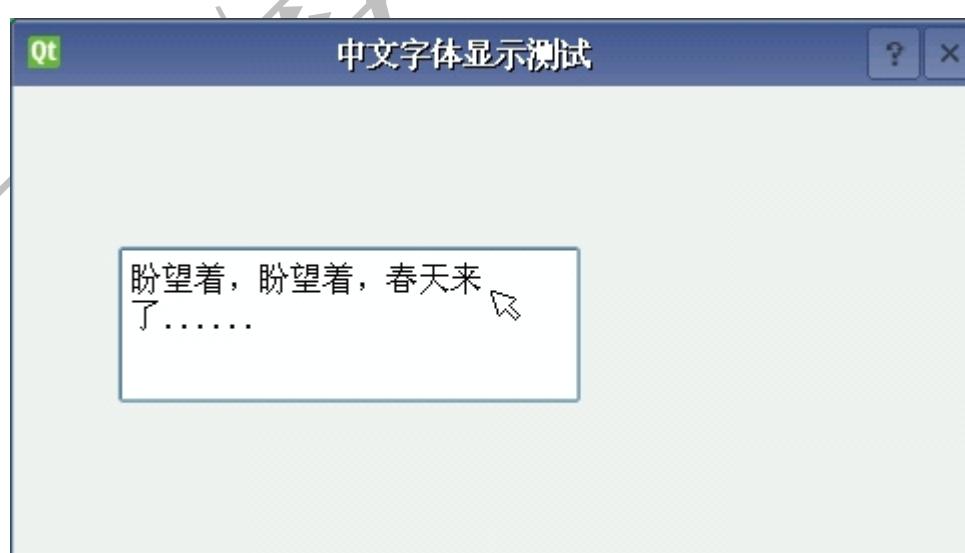
qt4-ChineseFontApp 源代码目录：

基础光盘\Linux-3.0.1\apptest\QT ApplicationTest\ChineseFontTest



The screenshot shows a terminal window titled "TT - 超级终端". The window has a menu bar with Chinese options: 文件 (File), 编辑 (Edit), 查看 (View), 呼叫 (Call), 传送 (Transfer), and 帮助 (Help). Below the menu is a toolbar with icons for file operations like copy, paste, and search. The main area displays a series of identical command-line prompts, each starting with "[root@FORLINX6410] #". The last prompt is highlighted with a red rectangle. At the bottom of the terminal window, there is a status bar showing connection information: 已连接 0:41:30 ANSIW | 115200 8-N-1 | SCROLL | CAPS | NUM | 捕 | 打印 | .

4.3 寸屏 16 号字体显示效果:



11-11 宿主机和开发板通过串口交互文件

11-11-1 开发板向 PC 机传送文件

点击超级终端菜单栏“发送→接收文件”，或在超级终端的空白处右击，在下拉菜单中选择接收文件



设置要存放文件的目录，点关闭。

在命令行上输入：

#sz /root/Documents/Demo1.jpg

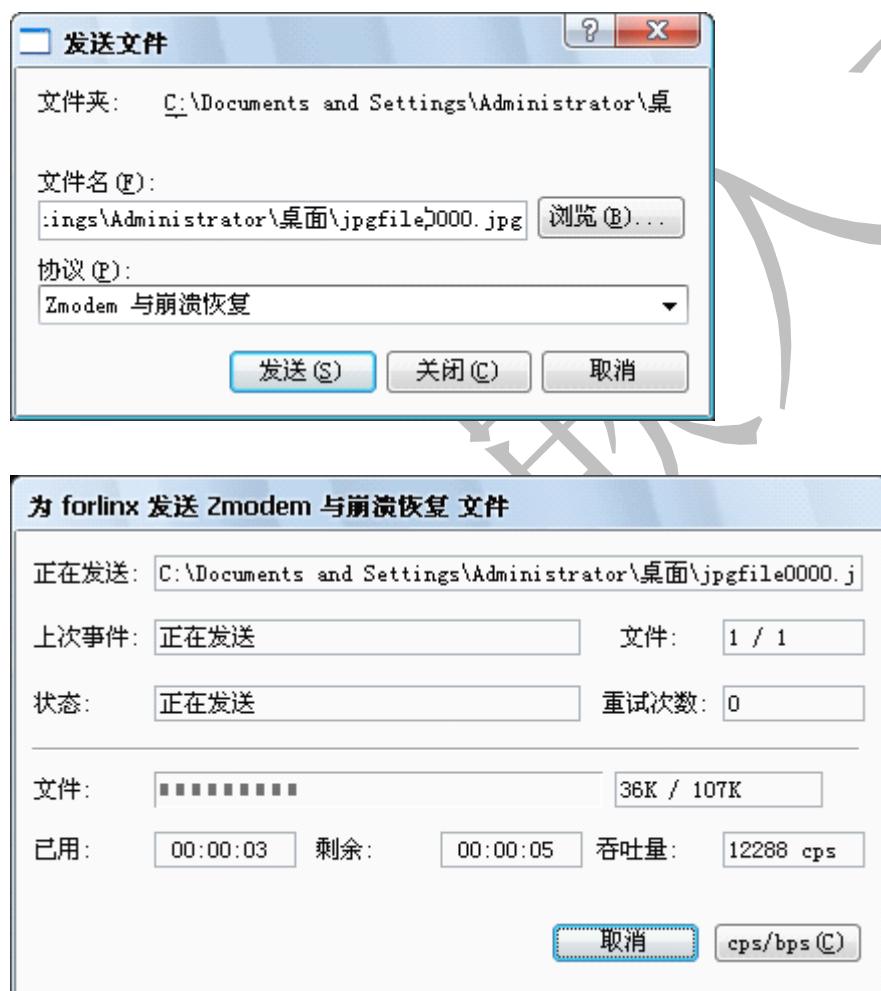


11-11-2 接收 PC 机传过来的文件

在Linux 命令行下进入要存放文件的目录, 然后在超级终端里面执行 rz 命令:

```
#cd /mnt  
#rz
```

在超级终端上点右键—>发送文件, 选择要发送的文件, 点击发送。传送成功后即可在命令行下看到该文件了。



11-12 TFTP 调试 Linux 内核

如果您在搞 Linux 内核开发，修改驱动后，通常需要使用 USB 的 DNW 功能把新编译的 zImage 放到开发板上面去，这是调试 Linux 内核的方法之一，另外一种快捷方便的调试方式就是在 Uboot 中使用 tftp 功能，开机自动从您的虚拟机获取内核文件加载到开发板上运行，再联合 NFS 功能，可以轻轻松松调试 Linux 内核和文件系统。

11-12-1 TFTP 服务在 Ubuntu12.04 上面的安装

TFTP 协议是简单文件传输协议，基于 UDP 协议，没有文件管理、用户控制功能。TFTP 分为服务器端程序和客户端程序，在主机上通常同时配置有 TFTP 服务端和客户端。

默认安装的 Ubuntu 系统没有包含 TFTP 的服务端和客户端，可以通过命令行来下载安装，步骤如下：

(1) 安装客户端。

```
root@forlinx:~# apt-get install tftp
```

(2) 安装服务端

```
root@forlinx:~# apt-get installtftpd
```

(3) 安装 inetd。

```
root@forlinx:~# apt-get installopenebsd-inetd
```

inetd 是监视一些网络请求的守护进程，其根据网络请求来调用相应的服务进程来处理连接请求。

(4) 在 "/" 目录下见一个 tftpboot, 把属性改成 777。

(5) 在 /etc/inetd.conf 里添加。

```
tftp dgram udp wait root /usr/sbin/in.tftpd/usr/sbin/in.tftpd -s /tftpboot
```

inetd.conf 是 inetd 的配置文件。inetd.conf 文件告诉 inetd 监听哪些网络端口，为每个端口启动哪个服务。

(6)重新加载 inetd 进程。

```
/etc/init.d/openbsd-inetd reload
```

(7)禁用防火墙。

```
ufw disable
```

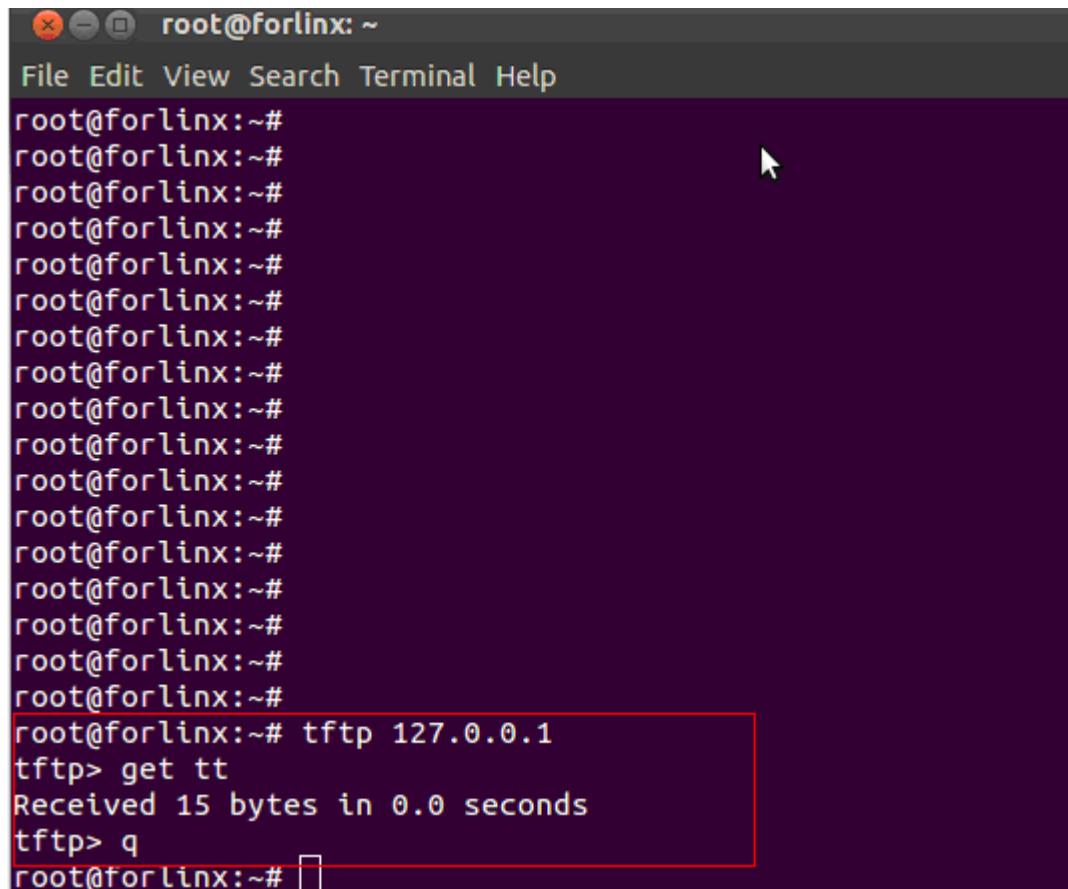
(8)测试 TFTP 服务器。

■ 从服务器下载文件:

1 登录服务器: tfpt 127.0.0.1

2 从服务器上获取文件: get 命令

如图:

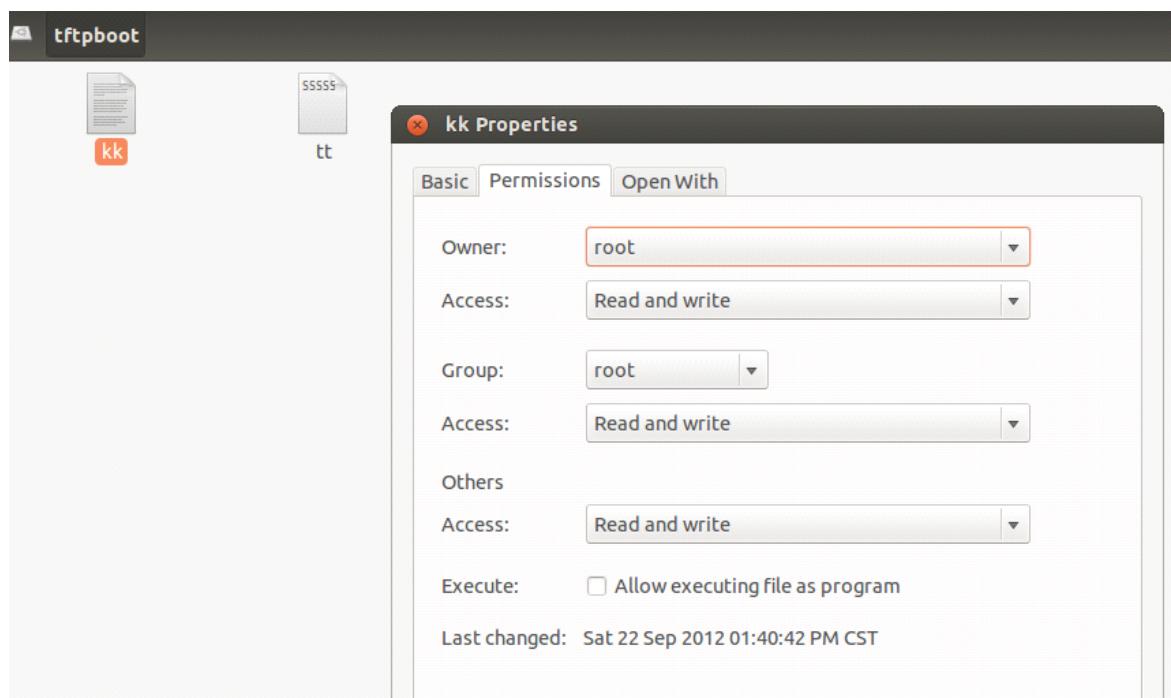


The screenshot shows a terminal window titled "root@forlinx: ~". The window has a dark background and light-colored text. It displays a series of blank command lines starting with "root@forlinx:~#". Then, a command is entered: "tfpt 127.0.0.1", which is highlighted with a red rectangle. After this, "tftp> get tt" is typed, followed by "Received 15 bytes in 0.0 seconds", and finally "tftp> q". The session ends with "root@forlinx:~#".

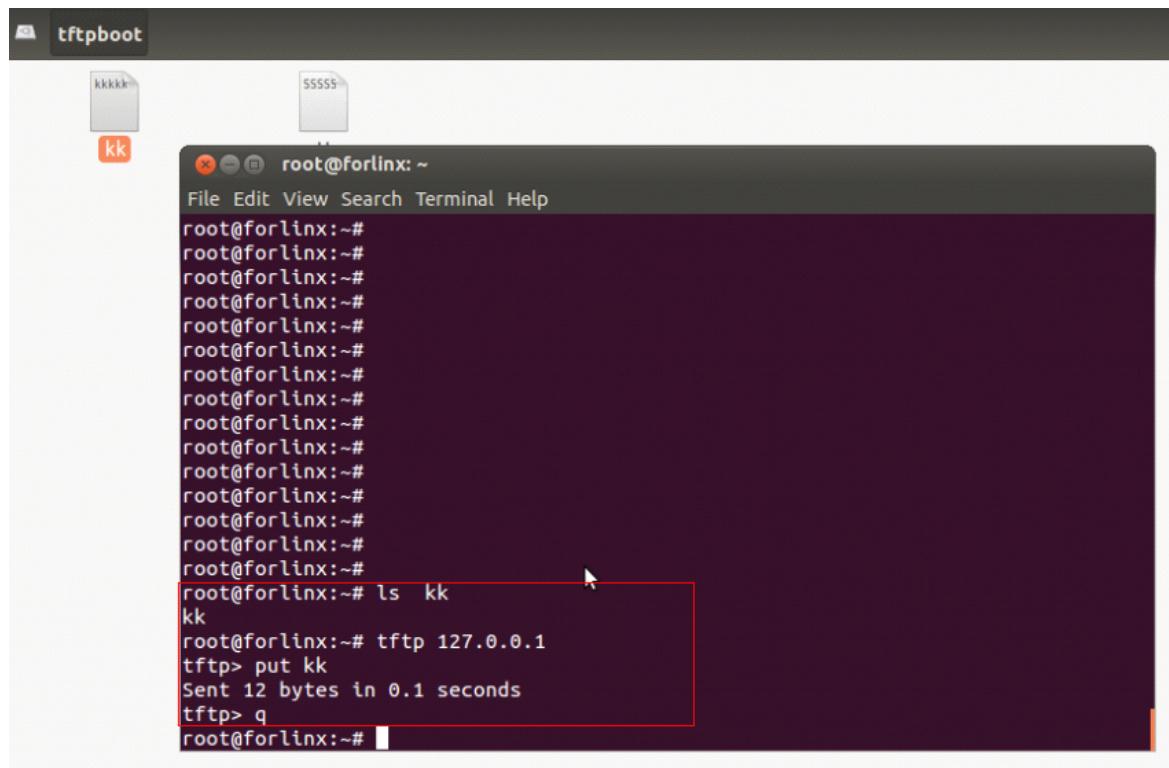
■ 上传文件到服务器：

上传文件时需要注意，在/tftpboot 下面要存在一个跟您上传文件名相同的文件，这个文件可以是空文件，但是该文件需要具备可读可写权限，否者会出现“Errorcode 2: Access violation”错误。

如图，上传文件前，建立相同文件名的空文件，及修改可读可写权限。



上传文件后，可以看到 kk 文件里面确实已经有新内容了，也就是说文件进行了覆盖。

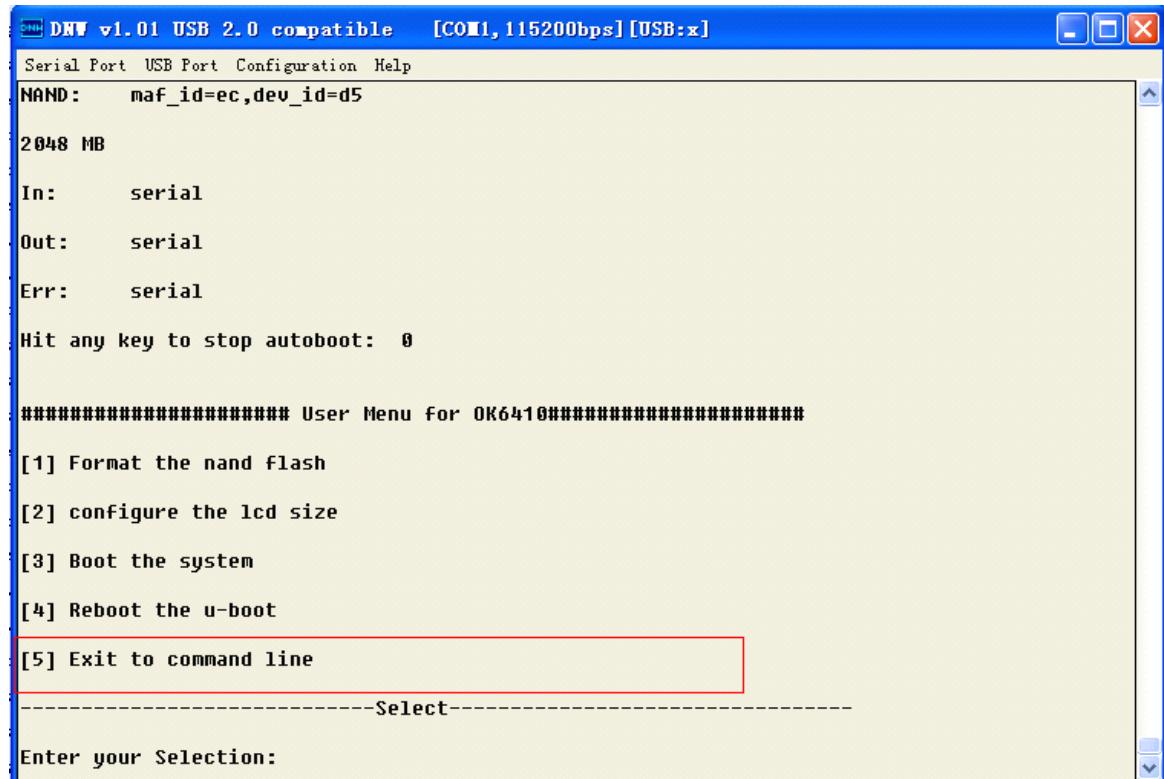


The screenshot shows a terminal window titled "tftpboot". Inside the terminal, there is a file browser window with two files: "kkkkk" and "55555". The terminal itself has the following command history:

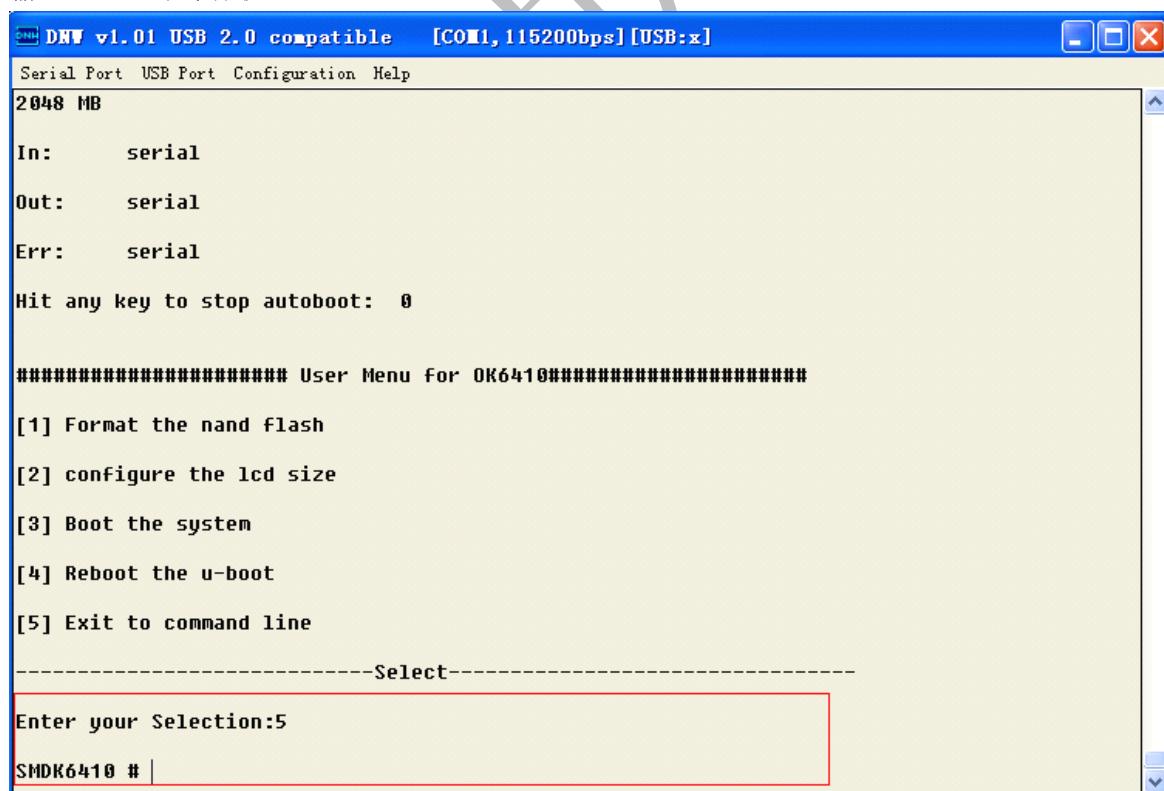
```
root@forlinx:~#  
root@forlinx:~# ls kk  
kk  
root@forlinx:~# tftp 127.0.0.1  
tftp> put kk  
Sent 12 bytes in 0.1 seconds  
tftp> q  
root@forlinx:~#
```

11-12-2 Uboot 设置 TFTP 启动内核

开发板与 PC 通过串口连接好，打开开发板的电源，按 PC 的空格键，进入 Uboot 菜单状态，



输入 5 进入命令行状态。



IP 地址设置：

```
# 设置开发板 IP 地址:  
setenv ipaddr 192.168.0.232  
# 设置虚拟机 IP 地址:  
setenv serverip 192.168.0.231
```

#TFTP 设置

setenv bootcmd tftp 20008000 zImage\; bootm 20008000

```
# 保存设置参数  
saveenv
```

把内核文件 **zImage** 放到/tftpboot 目录下面，重新启动开发板，Uboot 会自动从 **tftp** 服务端获取 **zImage** 文件了，以后更新 **zImage** 文件，只需要更新/tftpboot 目录下面的 **zImage** 即可。

下面的 Uboot 命令可以设置从 **NandFlash** 启动内核，开发板默认采用这种方式。

```
setenv bootcmd nand read C0008000 100000 500000\; bootm C0008000
```

我们提供了 **nfs-tftp.txt** 文件，里面有以上命令，您可以复制到 **uboot** 命令行状态。

另外需要注意一下虚拟机与您的宿主机之间网络采用桥接方式，如果采用 **NAT** 方式，开发板是无法连接到虚拟机的，采用 **TFTP** 前建议您使用 **ping** 命令检测开发板与虚拟机之间网络的状态。