mini2440+android 移植详解

潘应云（南京师范大学，QQ: 29548754，848682821,Email:panyingyun@gmail.com）

最近做了一些移植方面的事情，心想应该写一些东西下来，以便供更多的人参考。
第一章主要介绍了 Android 应用程序开发的环境搭建，并运行一个简单的 Hello world 程序。
第二章主要介绍了移植 Android 平台所需要的资料。
第三章主要介绍了移植 Android 平台的详细步骤。

目录

第一章　Android 开发环境搭建

目标：主要了解 Android 系统、学会 Android 环境搭建、熟悉在 Android 模拟平台上开发、调试简单的应用程序。

安装所需要的软件包下载地址：

Android SDK　　android-sdk-windows-1.1_r1.zip

下载：　http://code.google.com/android/download.html

Eclipse (Europa)　　eclipse-SDK-3.3.1.1-win32.zip

下载：　http://www.eclipse.org/downloads/

JDK6　　jdk-6u13-windows-i586-p.exe

下载：　http://java.sun.com/javase/downloads/index.jsp

交叉编译器：arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

下载地址：http://www.codesourcery.com/gnu_toolchains/arm/portal/release644

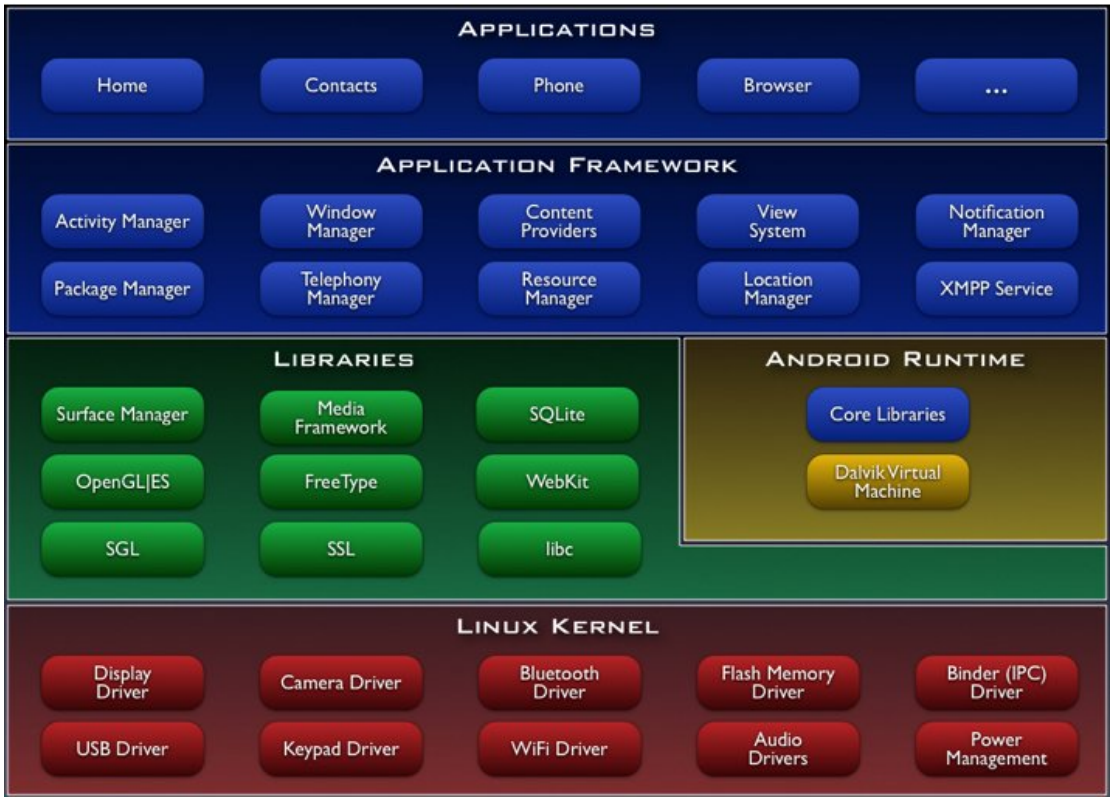
Android内核：linux-2.6.25-android-1.0_r1.tar.gz

下载地址：http://code.google.com/p/android/downloads/list

1、Android SDK 介绍

参考 1：http://www.androidin.com/pub/Android_Docs/Android_Docs_Trans/toolbox/index.html
参考 2：http://www.seesnow.net/archive/2009/03/08/ggandroidsjyykfhjddj.aspx
参考 3：http://bbs.hiqq.com.cn/10873-1-1/

令人激动的 Google 手机操作系统平台-Android 在 2007 年 11 月 13 日正式发布了，这是一个开放源代码的操作系统，内核为 Linux. 作为开发者，我们所关心的是这个平台的架构以及所支持的开发语言。下面是这个平台的架构模型：



这个平台有以下功能：

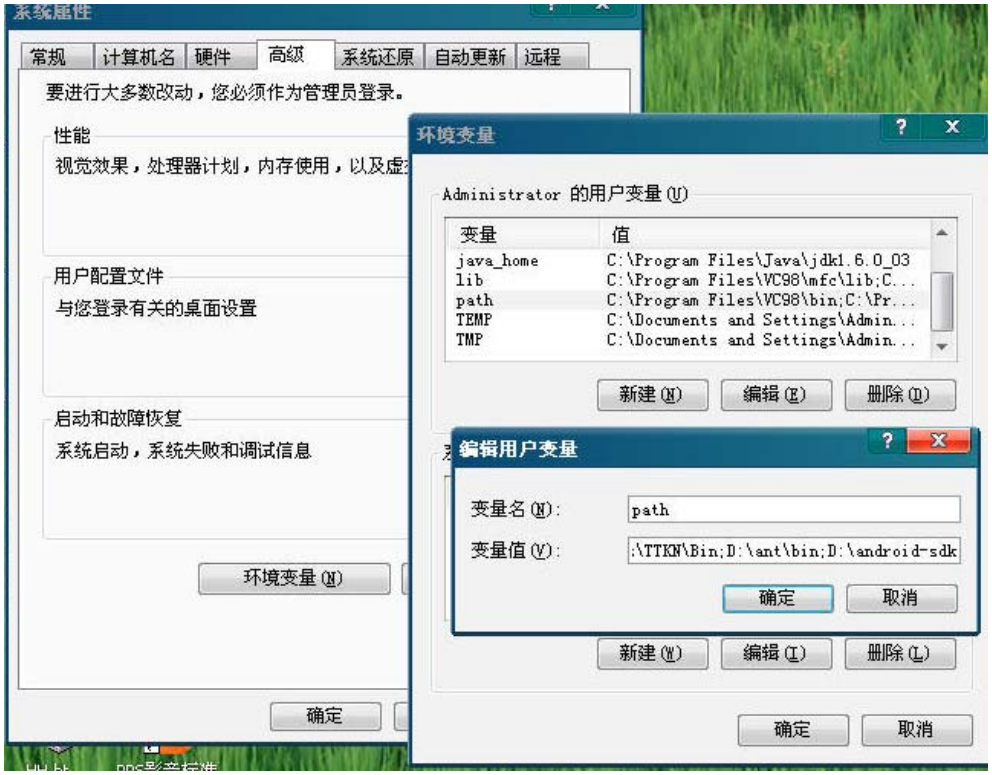
+ Application framework: 可重用的和可替换的组件部分，在这个层面上，所有的软件都是平等的。
+ Dalvik virtul machine: 一个基于 Linux 的虚拟机。
+ Integrated browser: 一个基于开源的 WebKit 引擎的浏览器，在应用程序层。
+ Optimized graphics: 包含一个自定义的 2D 图形库和基于 OpenGL ES 1.0 标准的 3D 实现。
+ SQLite: 数据库
+ Media support: 通用的音频,视频和对各种图片格式的支持(MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
+ GSM Telephony: GSM 移动网络, 硬件支持。
+ Bluetooth, EDGE, 3G, and WiFi: 都依赖于硬件支持。
+ Camera, GPS, compass, and accelerometer: 都依赖于硬件支持。
+ Rich development environment: 包含一套完整的开发工具集，方便跟踪调试，内存检测和性能测试，而且提供了 Eclipse 的插件。

2、Eclipse 中装载 Android 插件

下面我们就来亲身体验一下 Android 程序的开发之旅。
先介绍一下开发环境，下面是对系统及相关软件的版本要求：
操作系统：
Windows XP or Vista
Mac OS X 10.4.8 or later (x86 only)
Linux (tested on Linux Ubuntu Dapper Drake)

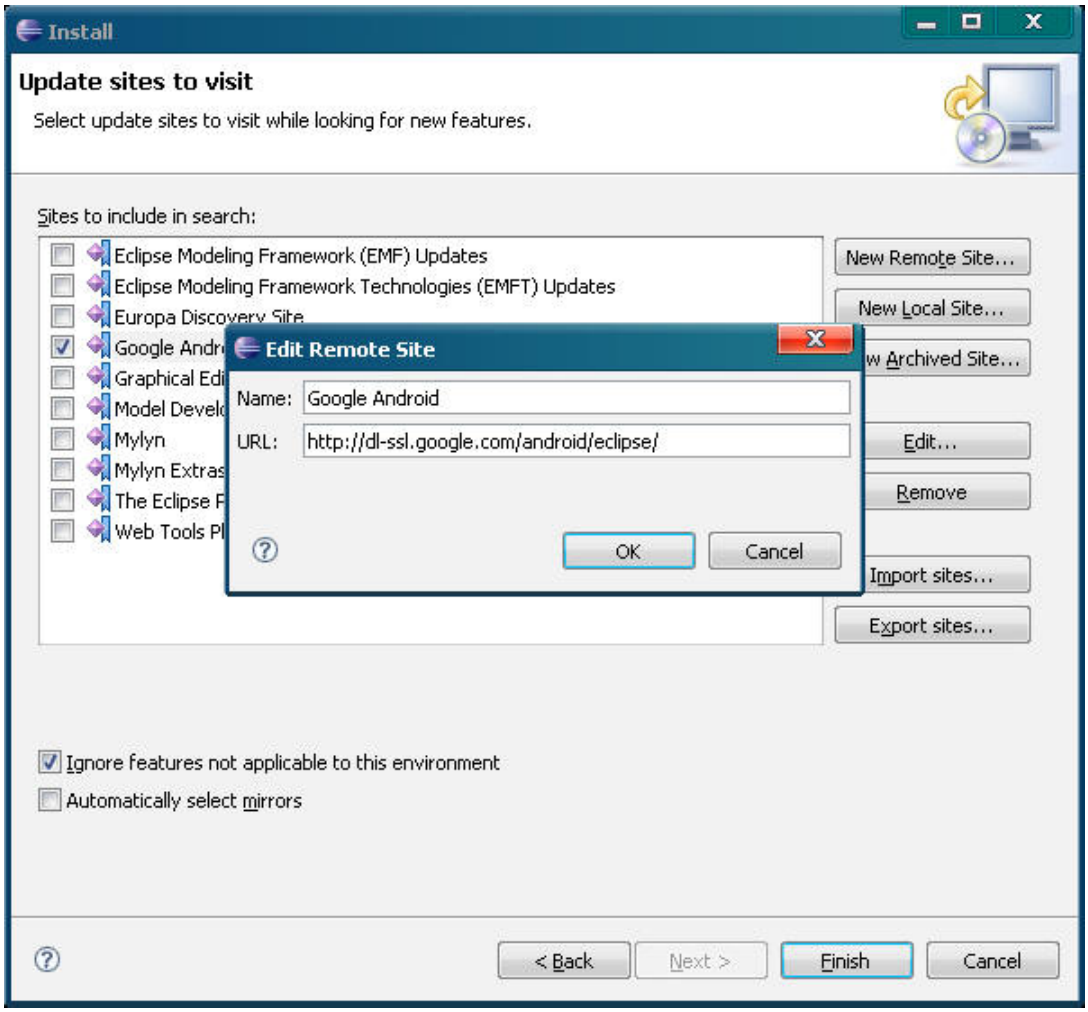
Supported Development Environments
Eclipse
Eclipse 3.2, 3.3 (Europa)
Android Development Tools plugin (optional)
Other development environments or IDEs
JDK 6 (JRE alone is not sufficient)
Not compatible with Gnu Compiler for Java (gcj)
Apache Ant 1.6.5 or later for Linux and Mac, 1.7 or later for Windows

我使用 Eclipse 3.3 + JDK 1.6. （+ Ant 1.7，这里暂时不用 Ant1.7 工具） 的组合。还有两个重要的就是: Android SDK 以及 Android 用于 Eclipse 中的插件。
Android SDK 的下载链接: http://code.google.com/android/
如果你是第一次使用这些软件，请注意安装顺序和设置好环境变量。 一般的顺序是先安装 JDK 然后 解压 ant 压缩包，然后设置 java 环境变量和 ant 环境变量， 然后是解压 Android SDK ，再设置 Android SDK 的环境变量。 总之就是把 JDK, ANT, Android SDK 的路径添加到 path 里。
环境变量设置：我的电脑－》属性－》高级－》编辑 path 变量



Android for eclipse plug in 在安装过程很简单,通过网络安装插件就可以了,这个是 URL: https://dl-ssl.google.com/android/eclipse/
然后是安装 Eclipse 的开发插件。
1）在 Eclipse 的菜单里选择 Software Updates > Find and Install...
2）在随后出现的窗口里选 Search for new features to install，然后“下一步”
3）点 New Remote Site
4）在这里随便给这个远程地址输入一个名字(比如 Google Android)，在下面输入网址

http://dl-ssl.google.com/android/eclipse/，然后点 OK，退回到上一级对话框后点“完成”。（注意：这里需要主机连接网络，有时候需要多试几次）

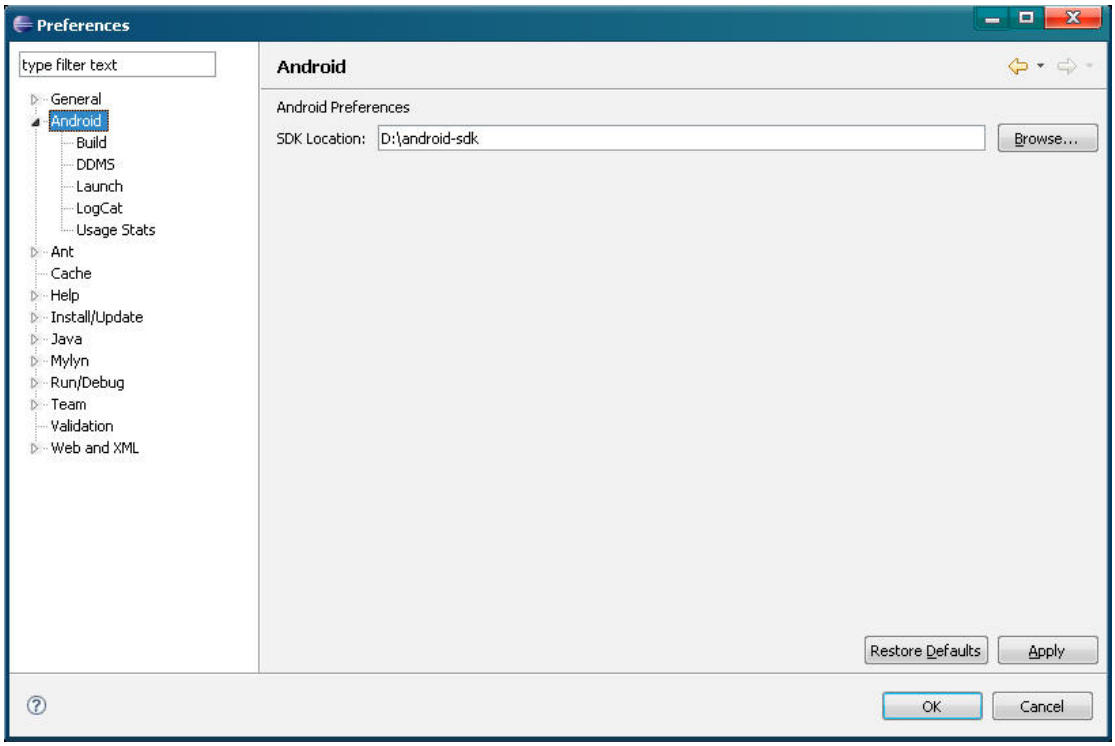


5）在随后结果窗口中，选择 Android Plugin > Eclipse Integration > Android Development Tools，然后“下一步”

6）接受 license 后再“下一步”，然后“完成”

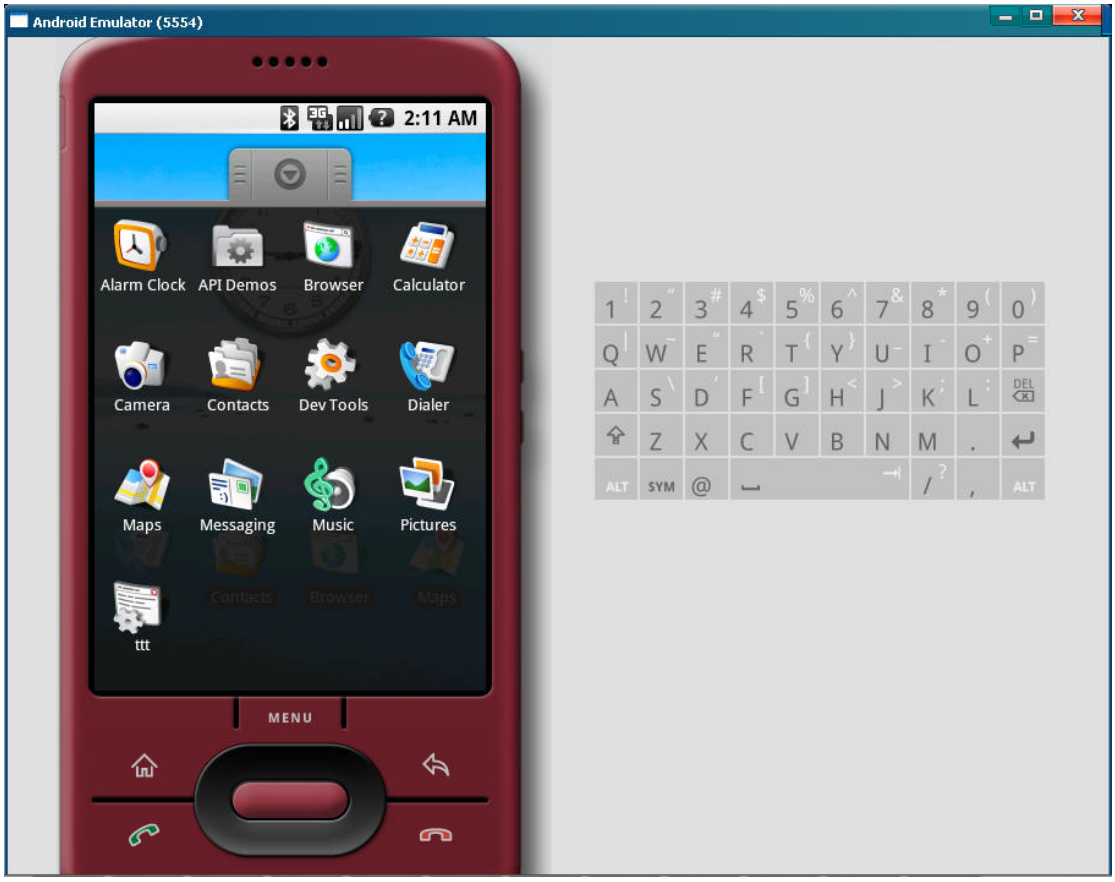
7）然后点 Install All，安装完后重启 Eclipse

8)重新启动 Eclipse 后，选择 Eclipse 菜单 Window -> Preferences -> 选择左侧的 Android 项，在右侧 SDK Location 项中输入你的 Android SDK 解压缩后的目录，



3、Android Emulator 使用

进入 Android sdk 目录，打开 tool 目录，双击运行 emulator.exe（时间比较长，需要慢慢等待）。



4、交叉编译工具链安装

交叉编译工具安装在 Linux 操作系统中，本人使用的是 RedHat9.0 版本，在其它 linux 版本中采用相似的。
在根目录（当然可以建立自己的工作目录）下进行
tar jxvf **arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2**
vi /root/.bashrc
在最后一行添加：
export PATH=$PATH:（你的目录路径）/arm-2008q3/bin
注销系统后，在终端中输入 arm-none-linux-gnueabi-gcc –v 有版本显示则表示设置 OK.

5、Android 的源代码结构与编译方法

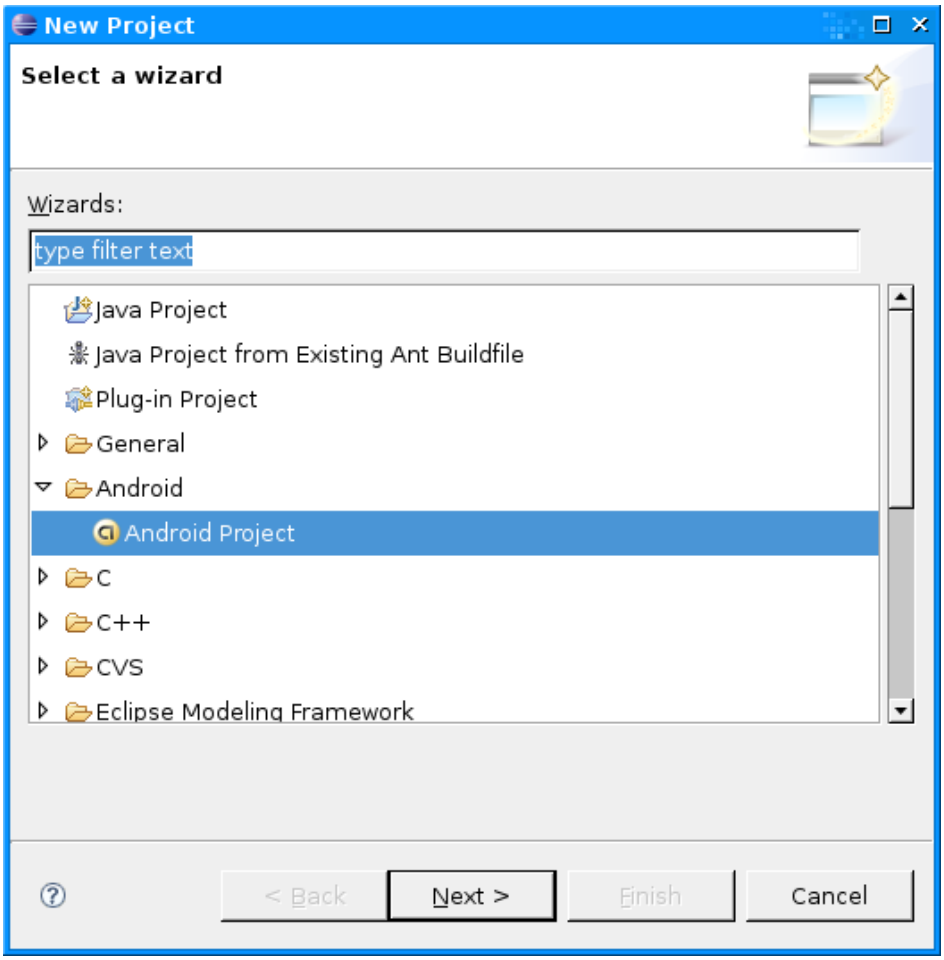
参见 Android SDK.

6、在模拟器上运行 Android 系统

进入 Android sdk 目录，打开 tool 目录，双击运行 emulator.exe。

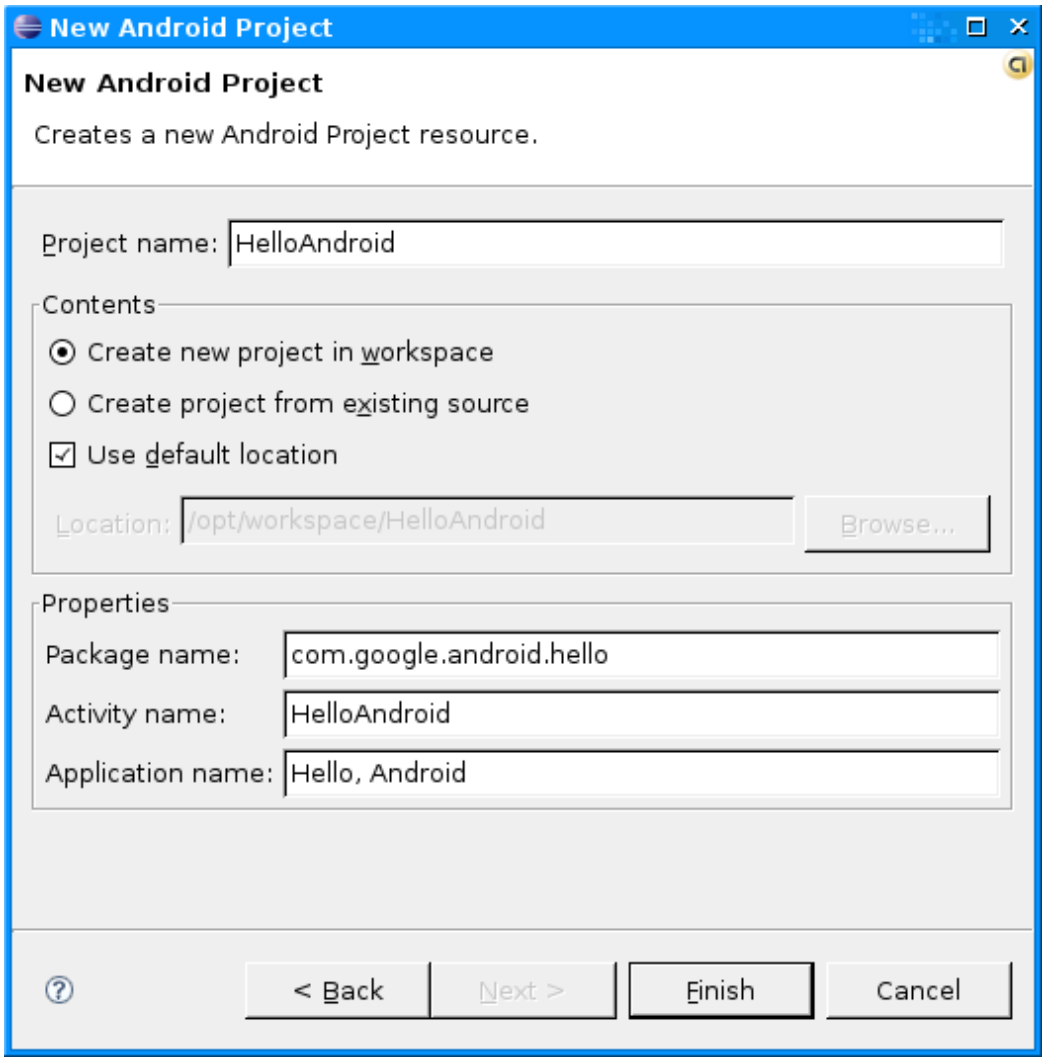
7、编写、调试、运行 Hello, Android!程序

创建一个项目:
创建一个新项目是很简单的，只要你安装了 Eclipse 插件，并且你的 Eclipse 软件版本在 3.2 或 3.3，你就可以开始开发了。
首先，看一下要创建"Hello, World"程序从高级层面上有哪些步骤：
1, 通过 File -> New -> Project 菜单，建立新项目"Android Project"
2, 填写新项目各种参数。
3, 编辑自动生成的代码模板。
尽此而已，我们通过下面的详细说明来完成每个步骤。
1， 创建一个新的 Android 项目
启动 Eclipse, 选择 File -> New -> Project 菜单, 如果你安装好了 Android 的 Eclipse 插件，你将会在弹出的对话框中看到"Android Project" 的选项。



选择“Android Project”,点击 Next 按钮。

2, 填写项目的细节参数.
下面的对话框需要你输入与项目有关的参数：



这个表格中详细介绍了每个参数的含义:
Project Name: 包含这个项目的文件夹的名称。
Package Name: 包名，遵循 JAVA 规范，用包名来区分不同的类是很重要的，例子中用到的是"com.google.android",你应该按照你的计划起一个有别于这个的路径的名称。
Activity Name: 这是项目的主类名，这个类将会是 Android 的 Activity 类的子类。一个 Activity 类是一个简单的启动程序和控制程序的类。它可以根据需要创建界面，但不是必须的。

Application Name: 一个易读的标题在你的应用程序上。
在"选择栏"的 "Use default location" 选项，允许你选择一个已存在的项目。

3，编辑自动生成的代码。
当项目创建后，你刚才创建的 HelloAndroid 就会是包含下面的代码.

```
public class HelloAndroid extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle)
```

```
    {
        super.onCreate(icicle);
        setContentView(R.layout.main);
    }
}
```

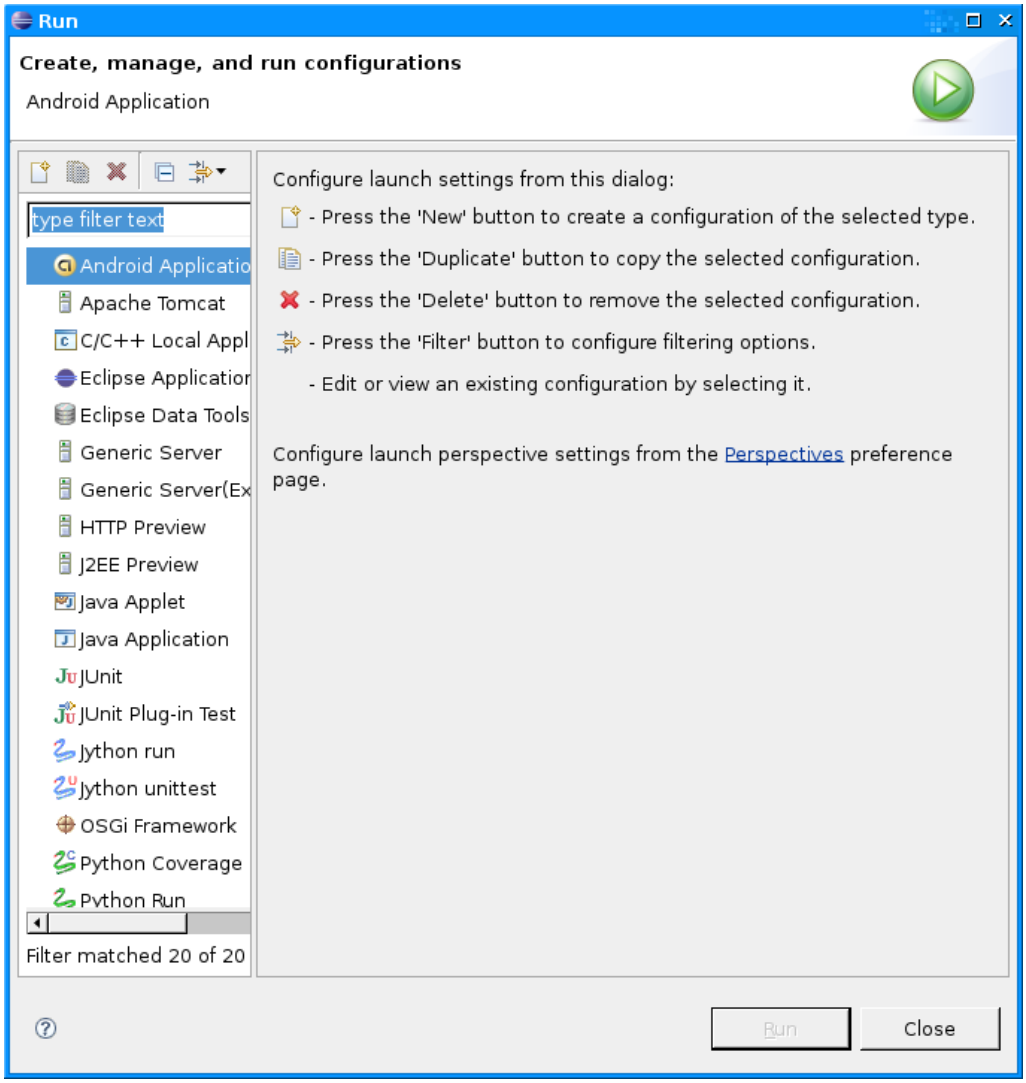
下面我们开始修改它

[构建界面]
当一个项目建立好以后，最直接的效果，就是在屏幕上显示一些文本，下面是完成后的代码，稍后我们在逐行解释。

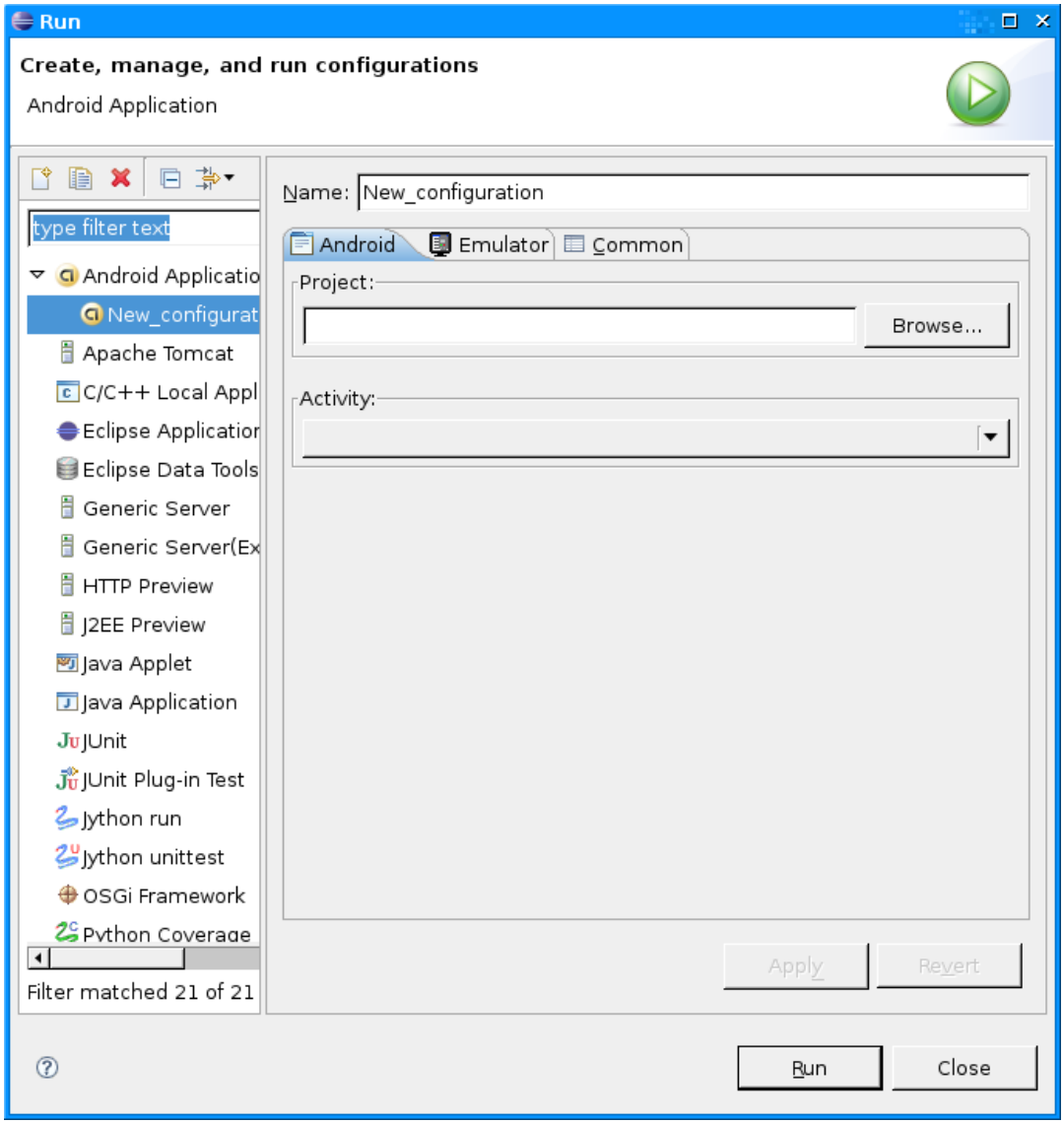
```
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

注意你还需要添加 import android.widget.TextView; 在代码开端处。
在 Android 程序中，用户界面是由叫做 Views 类来组织的。 一个 View 可以简单理解为可以绘制的对象，像选择按钮，一
个动画，或者一个文本标签(这个程序中)，这个显示文本标签的 View 子类叫做 TextView.
如何构造一个 TextView:
TexiView tv = new TextView(this);
TextView 的构造参数是 Android 程序的 Context 实例，Context 可以控制系统调用，它提供了诸如资源解析，访问数据库等
等。Activity 类继承自 Context 类，因为我们的 HelloAndroid 是 Activity 的子类， 所以它也是一个 Context 类，所以我们能用"this"在 TextView 构造中。
当我们构造完 TextView 后，我们需要告诉它显示什么:
tv.setText("Hello, Android");
这个步骤很简单，当我们完成了这些步骤以后，最后要把 TextView 显示在屏幕上。
setContentView(tv);
Activity 的 setContentView()方法指示出系统要用哪个 View 作为 Activity 的界面，如果一个 Activity 类没有执行这个方法，将会没有界面并且显示白屏。在这个程序中，我们要显示文本，所以我们传入已创建好的 TextView。
好了，程序代码已经写好，下面看看运行效果。
运行代码: Hello, Android

使用 Android 的 Eclipse 插件就可以很轻松的运行你的程序，选择 Run -> Open Run Dialog。你将会看到下面的对话框

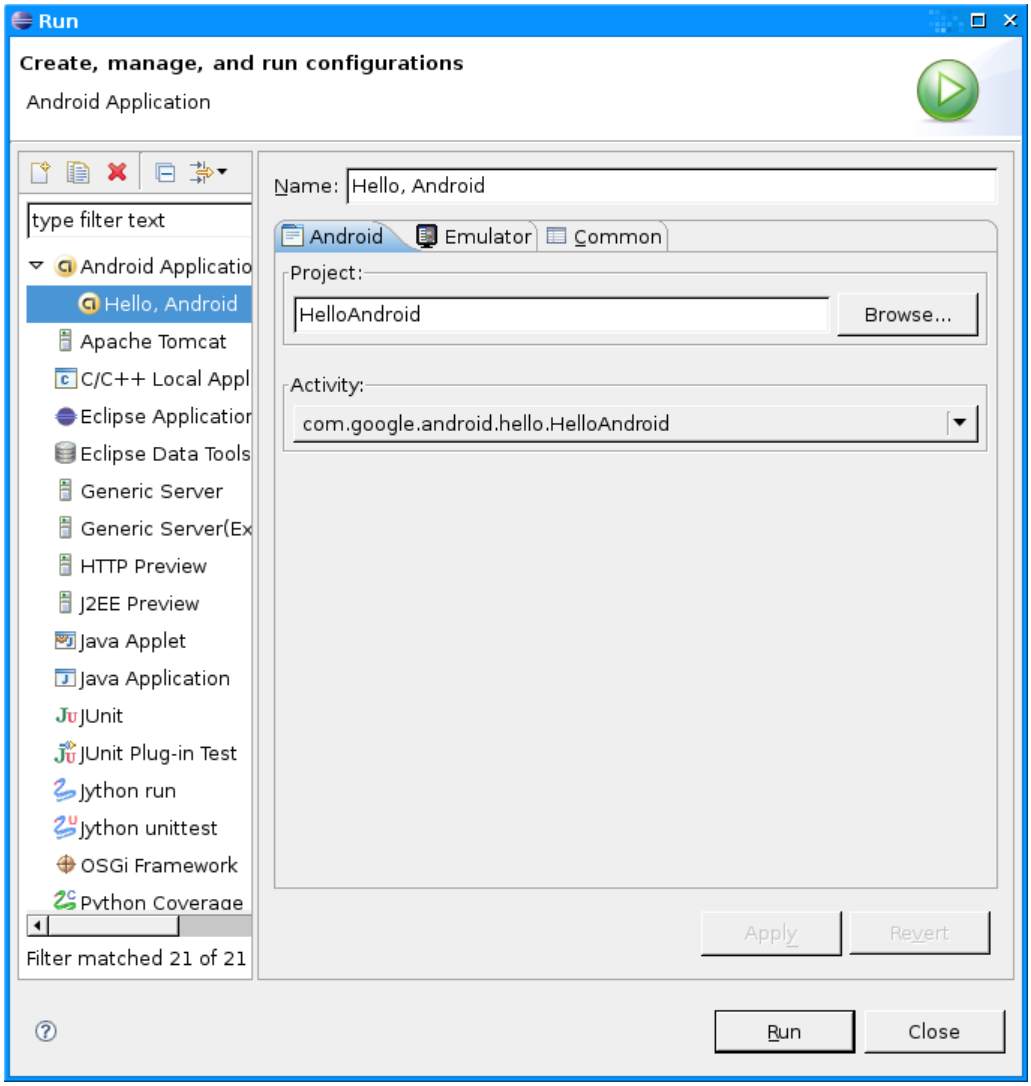


下一步， 高亮"Android Application" 标签，然后按下左上角的图标(就是像一片纸带个小星星那个)， 或者直接双击 "Android Application" 标签，你将会看到一个新的运行项目，名为"New_configuration".



取一个可以表意的名称，比如"Hello, Android"，然后通过 Browser 按钮选取你的项目(如果你有很多个项目在 Eclipse 中，确保你选择要运行的项目)，然后插件会自动搜索在你的项目中的 Activity 类并且将所有找到的添加在"Activity"标签的下拉列表中。 我们只有"Hello, Android"一个项目，所以它会作为默认选择。

点击"Apply" 按钮,下图



到这里，已经完成了，你只需要点击"Run"按钮，然后 Android 的模拟器将会启动，你的应用程序就会被显示出来（如果 Android 没有启动的话，这一步将比较慢，需要几分钟左右的时间）。

最后说明，你完全可以在 linux 平台下搭建 eclipse3.3＋Android SDK1.1＋JDK6.0 开发平台，过程基本一致。

第二章　Android 移植基础

目标：了解 Android 核心模块及相关技术、掌握移植要点、mini2440 硬件平台相关开发、bootloader 移植，为后面的 Android 移植打好基础。

1、Android 核心模块及相关技术

来源于http://www.nthcode.com/pubs/porting-android-to-a-new-device_zh.html

对于应用程序开发者来说，Android 建立在广受欢迎的开源 Eclipse 综合开发环境之上。开发者可以利用 Eclipse 写 Java 应用程序，然后汇编到 Java 字节码，之后由 Android 的 Dalvik 虚拟机（VM）解释并执行。

Google 声明他们的 Dalvik 虚拟机在速度和内存上与 Java 相比更有效，巧妙暗示了 Google 出于性能的原因写了自己的 VM。然而，在我们还没有看到任何支持数据之前，我们猜想 Google 创造 Dalvik 的真正原因是不想从 Sun 那里取得授权或者购买许可。Sun 是 Java 的创造者和管理者。

将 Java 翻译成 Dalvik 字节码，Google 就可以利用这个成熟的工具和庞大的 Java 开发群体。缺点是在 Android 手机上能够运行应用程序之前，开发者必须在编译的 Java 代码上运行 Google 翻译工具（然而 Google 将这一步骤在 Eclipse 插件中自动化了）。

Google 还提供 Android 模拟器。该模拟器是一个虚拟化的 ARM 微处理器，它运行与设备相同的系统代码和几乎一模一样的 Linux Kernel。Google 提供一个 Eclipse 插件以便于 Android 程序在模拟器中运行时能在 Eclipse 中编译调试。Android 模拟器是我们见过的最完整的模拟器，只要保证开发人员的电脑有足够的内存来同时运行 Eclipse 和模拟器就可以了。

Google 对 Kernel 做了哪些改动？

我们比较了一下 Android Kernel 和标准 Linux Kernel 的区别，发现 Google 变更了 75 个文件并增加了 88 个文件。我们准备了一个带注释的变更文件清单，附于文章末尾；在这里有一个简单总结。

Goldfish-44 个文件

Android 模拟器运行一个虚拟的 CPU，Google 叫这个 CPU 为 Goldfish。Goldfish 执行 ARM926T 的指令，并且有用于输入和输出的钩子，好比模拟器上的读取键和播放视频的输出键。

这些接口在定义 Goldfish 模拟器的文件中实现，并且不会被汇编到在真实设备上运行的 Kernel 中。所以我们可以放心的忽略这些文件。

YAFFS2-35 个文件

与 PC 在磁盘上存储文件不同，手机在固态闪存芯片上存储文件。HTC G1 用的是 NAND 闪存，一种因其高品质低价格而变得越来越流行的闪存。

YAFFS2 是“Yet Another Flash File System, 2nd edition（另一个闪存系统文件，第二版）的缩写，为 Linux Kernel 和 NAND 闪存设备提供高性能接口。YAFFS2 对 Linux 早就免费开放。然而，它不是标准 2.6.25 Linux Kernel 的一部分，所以 Google 将它添加到 Android 上。

蓝牙-10 个文件

Google 在蓝牙通讯协议栈中对 10 个文件进行了修改。修改的结果修正了与蓝牙耳机关联的 6 个明显错误，同时添加了蓝牙调试和访问控制功能。

调度-5 个文件

Android Kernel 同时包含了对 CPU 进程调度和时间维持算法的略微修改。我们还不知道关于这些修改的历史，但是其影响在粗略检查下并不明显。

新 Android 功能-28 个文件

除了改错和其他一些小变动外，Android 涵盖了一系列新的子系统值得一提，包括下列：

IPC Binder

IPC Binder 是一个进程间通信机制。它允许程序通过一套高端 API 设置向其他程序提供服务，而不是通过标准 Linux。网络搜索显示 Binder 概念始于 Be 公司，在 Google 为 Android 编写新 Binder 之前已经在 Palm 软件中实现了。

低内存杀手

Android 增加了一个低内存杀手，每次使用这个功能，扫描正在运行的程序，然后杀掉一个。在我们的粗略测试中不能清楚的显示为什么 Android 在标准 Linux Kernel 已有的低内存杀手之上又增加了一个。

Ashmem

Ashmem 是一个匿名共享内存（Anonymous SHared MEMory）系统，该系统增加了接口因此进程间可以共享具名内存块。举一个例子，系统可以利用 Ashmem 存储图标，当绘制用户界面的时候多个进程也可以访问。Ashmem 优于传统 Linux 共享内存表现在当共享内存块不再被用的时候，它为 Kernel 提供一种回收这些共享内存块的手段。如果一个程序尝试访问 Kernel 释放的一个共享内存块，它将会收到一个错误提示，然后重新分配内存并重载数据。

内存控制台和日志装置

Android 为调试错误增加了将 Kernel 记录信息储存到内存缓冲上的功能。另外，Android 还增加了单独日志模块，这样用户进程就可以读写用户日志信息了。

Android 调试桥

调试嵌入式设备可以说是个挑战。Google 创造了 Android 调试桥（简称 ADB）来简化调试，该调试通过一个USB连接运行Android的硬件设备和开发者用于编写应用程序的台式电脑之间。

Android 同时增加了一个新的实时时钟、一个交换支持和一个计时的 GPIO 支持。我们在文章结尾列举了影响这些新模块的文件。

电源管理-5 个文件

电源管理是一个好的移动设备中最难处理的问题之一，因此我们将其单独分成一组。有趣的是 Google 给 Linux 增加了一个新的电源管理系统，而不是利用现有的。我们在文章结尾列举了受影响的文件。

其他改变-36 个文件

除了上边列举的例子之外，我们还发现一系列“各式各样”得改变。和其他的改变相比，这些改变包括了额外调试支持、键盘灯控制和 TCP 网络管理。

网络过滤器-0 个文件

最后，我们的变更文件列表显示网络过滤器有 22 个被改变的文件。然而，测试显示只有文件名称得大小写不同（xt_DSCP.c 对应 xc_dscp.c），文件内容完全一致。因此在本文中我们将这些文件忽略不计。

2、Android 内核 Porting 要求

2.1 运行平台

Red Hat 9.0 平台(其它 linux 平台当然也可以的)

2.2 软件环境

Android SDK　　android-sdk-windows-1.1_r1.zip

下载：　http://code.google.com/android/download.html

Eclipse (Europa)　　eclipse-SDK-3.3.1.1-win32.zip

下载：　http://www.eclipse.org/downloads/

JDK6　　jdk-6u13-windows-i586-p.exe

下载：　http://java.sun.com/javase/downloads/index.jsp

Android 内核：linux-2.6.25-android-1.0_r1.tar.gz

下载地址：http://code.google.com/p/android/downloads/list

交叉编译器：arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

下载地址：http://www.codesourcery.com/gnu_toolchains/arm/portal/release644

3、mini2440 硬件平台移植相关部分分析（参考 S3C2440 手册以及 mini2440 平台手册）

3.1 处理器体系结构及指令集

3.2 时钟系统
3.3 存储系统
3.3 主要功能控制器介绍

4、bootloader 移植

采用原先的 supervivi。
bootloader 移植，当然也可以使用 U-boot。可以参考网页
http://blog.chinaunix.net/u2/75270/showart_1779196.html
它详细介绍了 u-boot2008.10 非 nand_leagcy 移植 mini2440。
http://blog.chinaunix.net/u1/34474/showart.php?id=400228
它介绍了移植 U－Boot.1.2.0 到友善之臂 SBC2440V4

第三章 移植 Android 内核到 mini2440 平台

目标：掌握在 mini2440 平台上移植 Android 的完整过程，精通移植 Android 的关键步骤。

1、移植标准 linux2.6.25 到 mini2440

移植参考：
http://androidok.com/bbs/dispbbs.asp?boardid=5&Id=21
http://www.androidin.com/bbs/viewthread.php?tid=2741&extra=page%3D1&page=1
http://androidok.com/bbs/dispbbs.asp?boardid=5&Id=21（S3c6410 平台Android移植）
下面我们一步一步来进行 Android 的移植（之前应该首先安装好交叉编译器，设置好 PATH 环境变量）。
（1）解压内核
新建一个工作目录/Android，将 linux-2.6.25-Android-1.0_r1.tar.gz，放到该目录下
#tar zxvf linux-2.6.25-Android-1.0_r1.tar.gz
此时在本目录下多了一个 kernel.git 目录，这就是 Android linux 内核。
（2）解压编译器和设置编译器路径的环境变量
将 arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gun.tar.bz2，拷贝到工作目录/Android 下面。
#tar jxvf arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gun.tar.bz2，得到编译器文件夹 arm-2008q3。
#vi /root/.bashrc
在最后加上 PATH=$PATH:/android/arm-2008q3/bin
保存退出，重启或者注销系统。
（3）修改 Makefile
进入 kernel.git 目录，Vi Makefile. 第 194 行，
修改
ARCH ?= $(SUBARCH)
CROSS_COMPILE ?=arm-eabi-
为
ARCH ?= arm
CROSS_COMPILE ?=arm-none-linux-gnueabi-
保存退出
（4）生成配置文件.config
make menuconfig（原先有一些默认选项，我们在默认选项上进行(添加)或者(去除)）
为了简单起见，把每一个修改的选项都列举下来，不是每个选项都是必须的，比较重要的选项我用粗体标明，仅供参考。
General setup->
[ ]support for paging of anonymous memory(swap) (NEW)（去除）
[ ]BSD Process Accounting （去除）
[ ]Initial RAM filesystem and RAMdisk (initramfs/initrd) support（去除）
[*]Enable Android's Shared Memory Subsystem(添加) **(重要的选项)**

[*]Activate markers(添加)

Enable loadable module support ->

[*]Module unloading(添加)

[*] Forced module unloading(添加)

[ ] Module Version support （去除）

[*] Source checksum for all modules(添加)

Enable the block layer->采用默认，不做修改

System Type->

ARM system type->**(重要的选项)**

(X)Samsung S3C2410,S3C2412,S3C2440,S3C2442,S3C2440, S3C2442, S3C2443(添加)

[*] S3C2410 DMA support(添加)

S3C2440 Machines->

[*]SMDK2440 (添加)

[*]SMDK2440 with S3C2440 CPU module (添加)

修改完成后，Support ARM920T processor 和 Support Thumb user binaries 会自动选上

Bus support->采用默认，不做修改

Kernel Features->

[*]Preemtible Kernel(添加)

[*]Use the ARM EABI to compile the kernel (添加)

[*] Allow old ABI binaries to run with this kernel (ExPERIMEAL) (添加)

Boot Options->采用默认，不做修改

Floating point emulation->

[*]NVFPE math emulation(添加)

Userspace binary formats->

[*] Kernel support for ELF binaries

<*>Kernel support a.out and ECOFF binaries

< >Kernel support for MISC binaries

Power management options->

[*] Legacy Power Management API(DEPRECATED) (添加)

[*] Power Management Debug Support(添加)

[ ] Suspend to RAM and standby（去除）

<*> advanced Power Management Emulation(添加)

Netwoking->

Networking options->

< >IP:multicasting （去除）

< >IP:advanced router（去除）

<*> IP:kernel level autoconfiguartion(添加)

<*>IP:BOOTP support(添加)

< >IP:tunneling(添加)（去除）

< >IP:GRE tunnels over IP（去除）

< >IP:TCP syncookie support(disable per default) （去除）

<>IP virtual server support（去除）

< >The IPv6 protocol（去除）

[ ]Network packet filtering framework(Netfilter)-> （去除）

< >Asynchronous Transfer Mode(ATM) （去除）
< > 802.1d Ethernet Bridging（去除）
< >802.1Q WLAN support（去除）
< >DEOnet support（去除）
< >The IPX protocol（去除）
< >Appletalk protocol support（去除）
< >VAN router（去除）
[ ]Qos and/or fair queueing->（去除）

< >IrDA (infrared) subsystem support->（去除）
< >RxRPC session sockets（去除）

Device Drivers->
Generic Driver Options->
<*> Userspace firmware loading support(添加)
<*>Memory Technology Device(MTD) support->(添加) **(重要的选项)**
[*] MTD partitioning support(添加)
[*] Direct char device access to MTD devices(添加)
<*> Caching block device access to MTD devices(添加)
<*> NAND Device Support->(添加)
<*>NAND Flash support for S3C2410/S3C2440 Soc(添加)
[*] S3C2410 NAND driver debug(添加)

< > Parallel port support（去除）
Block devices->
<*> Loopback device support(添加)
<*> Network block device support(添加)
SCSI device support →(在备份的内核中还没有修改过来)
<*>SCSI device support
<*>SCSI target support
[*] legacy /proc/scsi/ support
<*> SCSI disk support
其余选项可以不选
[ ] Multiple devices driver support(RAID and LVM)-> （去除）
[*] Network device support->
<*> Dummy net driver support(添加)
< > Bonding driver support （去除）
< > EQL(serial line load balancing)support
< > Universal TUN/TAP device driver support
[*] Ethernet(10 ro 100Mbit)-> **(重要的选项)**
<*>DM9000 support(添加)
USB Network Adapters->(下面所以选项去除)
[ ] van interfaces support->（去除）
[ ] ATM drivers ->（去除）
< > PPP(point-to-point protocol) support

```
    < >     SLIP(serial line) support
    < >     Network console logging support(EXPERIMENTAL)
< > ISDN support->（去除）
Input device support->
            (240) Horizontal screen resolution(修改括号里面的值)
            (320) Vertical screen resolution(修改括号里面的值)
        < > Joystick    interface（去除）
        <*>Event interface(添加)
        <*>Event debugging(添加)
            ***Input Device Drivers ***
        [*]Keyboards->
            <*> AT key board
            <*> sun Type 4 and Type 5 keyboard(添加)
            <*>DEC          (添加)
            <*>XT           (添加)
            <*>Newton       (添加)
            <*>Stowaway keyboard(添加)
            <*>GPIO Buttons(添加)
            < >Generic Input Event device for Goldfish（去除）
        [*]Mice->
            <*>    PS/2 mouse
            [*]      ecalax TouchKit PS/2 protocol extension(添加)
            <*>        serial mouse(添加)
            <*>        Apple USB Touchpad Support(添加)
            <*>        DEC VSXXX-AA/GA mouse and VSXXX-AB tablet(添加)
            <*>        GPIO mouse(添加)

        [*]Touchscreens->(添加)
    Character devices->
        <>HDLC line discipline support（去除）
        <>SDL RISCom/8 card support（去除）
        <>Specialix IO8+ card support（去除）
        [ ]    Stallion multiport serical support（去除）
        Serical drivers->(重要的选项)
            <*>Sansung S3C2410/S3C2440/S3C2442/S3C2412 Serial port support(添加)
            [*]     Support for console on S3C2410 serial port(添加)
        [ ]Legacy(BSD) PTY support（去除）
        < >IPM top-level message handler->（去除）
        < >/dev/nvram support（去除）
        < >Siemens R3964 line discipline（去除）
    <*>I2C support->(添加)
        <*> I2C device interface(添加)
                I2C Algorithms->
                    < > I2C bit-banging interfaces（去除）
```

```
                    < > I2C PCF 8584 interfaces（去除）
                I2C Hardwre Bus support->
                    <* >S3C2410 I2C driver
                Miscellaneous I2C Chip support->
                    < >EEPROM reader（去除）
                    < >Philips PCF8574 and PCF8574A（去除）
                    < >Philips PCF8591（去除）
            [*]I2C core debugging messages(添加)
            [*]I2C Algorithm debugging messages(添加)
            [*]I2C Chip debugging messages(添加)
< >Hardware Monitoring support->（去除）
[ ]Watchdog Timer Support->（去除）
    Graphics support->
        <*>support for frame buffer devices->
            [ ]Enbale Tile Blitting Support（去除）
            < *> S3C2410 LCD framebuffer support(添加)
        [ ]Backlight & LCD device support->（去除）
        Console display driver support->
            [ ]VGA text console（去除）
            <*> Framebuffer Console support(添加)
            [*] Select compiled-in fonts(添加)
                [*]VGA8x16 font(添加)
          [*]Bootup logo->(添加)
    sound->采用默认，不做修改（后面移植声卡的时候再进行修改）
    [*]HID Devices->(子项全部选上)
    [*]USB support->
        <*> support for Host-side USB
        [ ] USB device filesystem（去除）
        <*> OHCI HCD support(添加)
        < *>USB Mass Storage support
        < > USB Modem(CDC ACM) support （去除）
        < >USB Printer support（去除）
        [ ]Datafab Compact Flash reader support（去除）
        [ ]Freeecom USB/ATAPI Bridge support（去除）
        [ ]ISD-200 USB/ATA Bridge support（去除）
        [ ]Microtech/ZiO!CompactFlahs/SmartMedia support（去除）
        [ ]SanDisk SDDR-09(and other SmartMedia)support（去除）
        [ ]SanDisk SDDR-55  SmartMedia support（去除）
        [ ]Lexar Jumpshot Compact Flash Reader（去除）
        < >USB Mustek MDC800 Digital Camera support(EXPERIMENTAL) （去除）
        < >Microtek X6USB scanner support（去除）
        [ ]USB Monitor（去除）
        [ ]USB Serial Converter support（去除）
        < >USB Auerswald ISDN support（去除）
```

< >USB Diamond Rio500 support（去除）
< >USB LCD driver support（去除）
<*>USB Gadget Support->
USB Peripheral Controller(Renesas M6692 USB Peripheral Controller)
(回车后，选择) （X）S3C2410 USB Device Controller(添加) **(重要的选项)**
[*] S3C2410 udc debug messages
<*>USB Gadget Drivers (添加)
<*>MMC/SD card support->(添加)
<*>Real Time Clock->
<*> Samsung S3C series SoC RTC(添加) **(重要的选项)**
Android->**(重要的选项)**
[*] RAM buffer console(添加)
[*]Android power driver(添加)

File systems->
[*]Ext2 extended attributes(添加)
< >Ext3 journalling file system support（去除）
[ ] Ext3 extended attibutes（去除）
[ ] Ext3 POSIX Access Control Lists（去除）
[ ] Reiserfs support （去除）
[ ] JFS filesystem support（去除）
[ ] Quota support（去除）
CD-ROM/DVD Filesystem->（子项全部去除）
DOS/FAT/NT filesystem->
<*>MSDOS fs support(添加)
<*>VFAT (Windows-95) fs support(添加)
<*>NTFS file system support(添加) **(重要的选项)**
<*>NTFS debugging support(添加)
<*>NTFS write support(添加) **(重要的选项)**
Miscellaneous filesystem->
< >Apple Macintosh file system support（去除）
< >BeOS file system(BeFS) support(read only) （去除）
< >BFS file system support（去除）
<*>YAFFS2 file system support(添加) **(重要的选项)**
<*> Lets Yaffs do its own ECC(添加) **(重要的选项)**
<*>Compressed ROM File system support(cramfs) (添加)
< >Minix File sytem support（去除）
< >ROM file system support（去除）
< >System V/Xenix/V7/Coherent file system support（去除）
< >UFS file system support（去除）
Network File Systems->
<*> NFS File system support(添加) **(重要的选项)**
<*> Provide NFSv4 client support(添加)
< > NFS server support（去除）

<* >Root file system on NFS(添加)
< >SMB file system support（去除）
< >NCP file sytem support（去除）
< >Coda file system support（去除）
< >Andrew File System support(AFS)（去除）

Partition Types->
[ ]Alpha OSF partition support（去除）
[ ]Macintosh partition table support（去除）
[ ]BSD disklabel(FreeBSD partition tables) support（去除）
[ ]Minix subpartition support（去除）
[ ]Solaris(x86) partition support（去除）
[ ]Unixware slices support（去除）
[*]Windows Logical Disk Manager(Dynamic Disk) support(添加)
[ ]SGI partition support（去除）
[ ]Sun partition tables support（去除）

Native language support->(选择以下语言，其它的不选)
<*> Codepage 437(United States,Canada) (添加)
<*>ASCII(United States) (添加)
<*>NLS ISO 8895-1(Latiin 1; Western European Language) (添加)
<*>Korean charset(CP949,EUC-KR) (添加)
<*>NLS UTF-8 (添加)

Kernel hacking->
[ ] Magic SysRq key（去除）
[ ] Kernel debugging（去除）
[*] Sample Kernel code->(添加)

Security options->(采用默认)
Cryptographic API->(采用默认)
Library routines->
[*]CRC-CCITT functions(添加)

到此，config 配置已经基本完成，在后面进行驱动修改时，我们还需要进行一些简单修改。

（5）编译生成 zImage

make zImage

出现错误 1：

```
  CC      arch/arm/plat-s3c24xx/s3c244x.o
arch/arm/plat-s3c24xx/s3c244x.c: In function 's3c244x_init_clocks':
arch/arm/plat-s3c24xx/s3c244x.c:121: error: implicit declaration of function 's3
c2410_baseclk_add'
make[1]: *** [arch/arm/plat-s3c24xx/s3c244x.o] Error 1
make: *** [arch/arm/plat-s3c24xx] Error 2
[root@localhost kernel.git]#
```

kernel.git/arch/arm/plat-s3c24xx/s3c244x.h 中加入 extern int s3c2410_baseclk_add(void);

参考出处：http://www.diybl.com/course/6_system/linux/Linuxjs/2008829/138614.html

重新编译，成功

（6）现在参考默认的 SMDK 开发板体系进行修改。

修改 kernel.git/arch/arm/tools/mach-types

第 379 行：

s3c2440　　ARCH_S3C2440　　S3C2440　　362

修改为：

s3c2440　　ARCH_S3C2440　　S3C2440　　782

各项的解释如下：

＃machines_is_XX　　CONFIG_XXX　　MACH_TYPE　　number

注意 number 不能与编译到内核中的其它开发板体系的编号冲突，我这里其实有冲突，但是最后好像没有负面影响。

注：内核启动时，是通过 bootloader 传入 MACH_TYPE 的编号来确定应该启动哪种开发板体系，没有用到的开发板体系，内核也会把它编译进去。所以，如果想要内核小点，在配置内核时，应该将内核中不用的开发板体系配置为不编译。mini2440 开发板 MACH_TYPE＝782，所以我们这里也用 782，如果用其它的值也是可以的，只要你改变以下 bootloader 的 MACH_TYPE 值就可以了。

（7）修改 machine 名称（当然也可以不修改）

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

找到 MACHINE_START(S3C2440, “SMDK2440”)。“SMDK2440”可以改成自己想要的名字，这样原来显示 SMDK2440 的地方就变成显示你自己定义的字符串；当然也可以不用修改。

（8）修改时钟

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

```
static void __init smdk2440_map_io(void)
{
    。。。。。。
    s3c24xx_init_clocks(12000000);    //修改，原来是 s3c24xx_init_clocks(16934400);
    。。。。。。
}
```

（9）修改背光

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

（注：这里修改背光的办法比较简单，比较正规的改法是添加一个管理背光的驱动程序。）

```
static void __init smdk2440_machine_init(void)
{
    s3c24xx_fb_set_platdata(&smdk2440_fb_info);
    platform_add_devices();
    s3c2410_gpio_cfgpin(S3C2410_GPG4,S3C2410_GPG4_OUTP); //添加
    s3c2410_gpio_setpin(S3C2410_GPG4,1);                 //添加
    smdk_machine_init();
}
```

（10）LCD 驱动移植

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

内核已经包含了对 S3C2440 支持的 LCD 驱动，只要在内核中选择

Support for frame buffer devices

Console display driver support

Bootup logo

以上三项启动选中，启动时才会出现小企鹅。

除此之外，还要对 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c 中的

smdk2440_lcd_cfg __initdata 结构体内进行设置，这里用的是 NEC3.5 英寸屏，320x240，应该对它进行设置

```
static struct s3c2410fb_display smdk2440_lcd_cfg __initdata ={


。。。。。。
        .pixclock = 100000, //  修改 ， 原来为 166667
。。。。。。
        .right_margin = 37, //修改 ， 原来为 8
        .hsync_len = 6,    //修改 ， 原来为 4
        .upper_margin =2, //修改 ， 原来为 8
        .lower_margin = 6, //修改 ， 原来为 7
        .vsync_len =2, //修改 ， 原来为 4
    };

static struct s3c2410fb_mach_info smdk2440_fb_info __initdata  ={
。。。。。。
.default_display =0
//add start
    .gpccon         = 0xaa955699,
    .gpccon_mask = 0xffc003cc,
    .gpcup          = 0x0000ffff,
    .gpcup_mask   = 0xffffffff,
    .gpdcon         = 0xaa95aaa1,
    .gpdcon_mask = 0xffc0fff0,
    .gpdup          = 0x0000faff,
    .gpdup_mask   = 0xffffffff,
// add end

.lpcsel = 0xf82, //修改 ， 原来为((0xCE6) &~7) | 1<<4,
};
```

（11）RTC 驱动移植

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

由于 kernel.git 内核自带了 RTC 驱动，但是在 mach-s3c2440.c 中没有激活。只要在结构体 struct platform_device *smdk2440_devices[] __initdata 数组中加入 RTC 结构体就可以了，

```
    &s3c_device_rtc,
```

该结构体定义在 kernel.git/arch/arm/plat-s3c24xx/devs.c 中。

（12）Nand Flash 驱动移植，修改文件

kernel.git/arch/arm/plat-s3c24xx/common-smdk.c

修改文件 kernel.git/arch/arm/plat-s3c24xx/common-smdk.c

第一，修改分区信息：

```
static struct mtd_partition smdk_default_nand_part[]={
static struct mtd_partition smdk_default_nand_part[] = {
```

```
        [0] = {
        .name = "bootloader",
        .offset = 0x00000000,
        .size = 0x00030000,
        },
        [1] = {
        .name = "kernel",
        .offset = 0x00050000,
        .size = 0x00200000,
        },
        [2] = {
        .name = "root",
        .offset = 0x00250000,
        .size = 0x03dac000,
        }
};
}
```

第二，再修改 s3c2410_platform_nand_smdk_nand_info smdk_nand_info = {

```
。。。。。。
.tacls = 0,
.twrph0 = 30,
.twrph1=0,
。。。。。。
};
```

（13）编译下载到板子运行进行测试

make zImage 得到 zImage 下载到开发板进行测试，可以看到小企鹅。启动信息如下：

```
Copy linux kernel from 0x00050000 to 0x30008000, size = 0x00200000 ... done
zImage magic = 0x016f2818
Setup linux parameters at 0x30000100
linux command line is: "console=ttySAC0 root=nfs nfsroot=192.168.1.100:/nfs_root
ip=192.168.1.70 init=/init"
MACH_TYPE = 782
NOW, Booting Linux.....
Uncompressing

Linux.....

done, booting the kernel.
Linux version 2.6.25 (root@localhost.localdomain) (gcc version 4.3.2 (Sourcery G++ Lite
2008q3-72) ) #4 PREEMPT Tue Apr 14

21:50:56 CST 2009
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: PanYingYunSMDK2440
```

ATAG_INITRD is deprecated; please update your bootloader.
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C244X: core 405.000 MHz, memory 101.250 MHz, peripheral 50.625 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: console=ttySAC0 root=nfs nfsroot=192.168.1.100:/nfs_root ip=192.168.1.70 init=/init
irq: clearing subpending status 00000003
irq: clearing subpending status 00000002
PID hash table entries: 256 (order: 8, 1024 bytes)
timer tcon=00000000, tcnt a4ca, tcfg 00000200,00000000, usec 00001e57
Console: colour dummy device 80x30
console [ttySAC0] enabled
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 60848KB available (3460K code, 435K data, 100K init)
SLUB: Genslabs=12, HWalign=32, Order=0-1, MinObjects=4, CPUs=1, Nodes=1
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
net_namespace: 540 bytes
android_power_init
android_power_init done
NET: Registered protocol family 16
S3C2410 Power Management, (c) 2004 Simtec Electronics
S3C2440: Initialising architecture
S3C2440: IRQ Support
S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
DMA channel 0 at c4800000, irq 33
DMA channel 1 at c4800040, irq 34
DMA channel 2 at c4800080, irq 35
DMA channel 3 at c48000c0, irq 36
S3C244X: Clock Support, DVS off
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 8
NET: Registered protocol family 20
NET: Registered protocol family 2

IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NetWinder Floating Point Emulator V0.97 (double precision)
ashmem: initialized
NTFS driver 2.1.29 [Flags: R/W DEBUG].
yaffs Apr 14 2009 18:58:12 Installing.
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
Console: switching to colour frame buffer device 30x20
fb0: s3c2410fb frame buffer device
s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
brd: module loaded
loop: module loaded
nbd: registered device at major 43
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c2440-nand s3c2440-nand: Tacls=1, 9ns Twrph0=4 39ns, Twrph1=1 9ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
Scanning device for bad blocks
Creating 3 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00000000-0x00030000 : "Bootloader(Panyingyun)"
0x00050000-0x00250000 : "Kernel(panyingyun)"
0x00250000-0x03dac000 : "User(panyingyun)"
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
s3c2410_udc: debugfs dir creation failed -19
mice: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2410-rtc s3c2410-rtc: rtc disabled, re-enabling
s3c2410-rtc s3c2410-rtc: rtc core: registered s3c as rtc0
i2c /dev entries driver
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 98 KHz
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter

logger: created 64K log 'log_main'
logger: created 64K log 'log_events'
logger: created 64K log 'log_radio'
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-04-14 21:48:06 UTC (1239745686)

（14）网卡 DM9000 驱动移植
1、拷贝dm9000x.c到 /kernel.git/driver/net/下面，参考附件dm9000x.c
2、修改/kernel.git/driver/net/Makefile 文件，
obj-$(CONFIG_DM9000)　　+=dm9000.o 修改为
obj-$(CONFIG_DM9000)　　+=dm9000x.o
3、修改 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c
在开头部分添加下面宏定义
#define pSMDK2410_ETH_IO 0x19000000
#define vSMDK2410_ETH_IO 0xd0000000
#define SMDK2410_ETH_IRQ IRQ_EINT9
修改 static struct map_desc smdk2440_iodesc[] __initdata = {

```
    /* ISA IO Space map (memory space selected by A24) */
        /*
    {
        .virtual    = (u32)S3C24XX_VA_ISA_WORD,
        .pfn        = __phys_to_pfn(S3C2410_CS2),
        .length         = 0x10000,
        .type       = MT_DEVICE,
    }, {
        .virtual    = (u32)S3C24XX_VA_ISA_WORD + 0x10000,
        .pfn        = __phys_to_pfn(S3C2410_CS2 + (1<<24)),
        .length         = SZ_4M,
        .type       = MT_DEVICE,
    }, {
        .virtual    = (u32)S3C24XX_VA_ISA_BYTE,
        .pfn        = __phys_to_pfn(S3C2410_CS2),
        .length         = 0x10000,
        .type       = MT_DEVICE,
    }, {
        .virtual    = (u32)S3C24XX_VA_ISA_BYTE + 0x10000,
        .pfn        = __phys_to_pfn(S3C2410_CS2 + (1<<24)),
        .length         = SZ_4M,
        .type       = MT_DEVICE,
    }*/
```

```
    {vSMDK2410_ETH_IO, pSMDK2410_ETH_IO,SZ_1M,MT_DEVICE},
};
```

这个其实就是去掉原来的定义，增加了一个网口的地址。

4、配置 网卡驱动 选中 dm9000

```
    make menuconfig
Device Drivers->
    [*]Network device support->
        [*]Ethernet(10 or 100Mbit)->
            <*>DM9000 support
```

可以进行以下简单的测试：

NFS 启动（测试成功）：

"console=ttySAC0 root=nfs nfsroot=192.168.1.100:/nfs_root ip=192.168.1.70 init=/init

Yaffs 启动(测试成功)：

"noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0

或者

"noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0 rootfstype=yaffs

或者

"noinitrd root=31:02 init=/linuxrc console=ttySAC0 rootfstype=yaffs

可以使用《mini2440 开发板之 Android 使用手册》的文件系统先进行测试，发现可以 NFS 启动了,可以看到 google 的大钟了。

（15）触摸屏移植

1、拷贝 s3c2410_ts.c 到 kernel.git/driver/input/touchscreen/

2、修改 kernel.git/driver/input/touchscreen/Makefile,在第六行添加下面一句

```
  obj-$(CONFIG_TOUCHSCREEN_S3C2410)  += s3c2410_ts.o
```

3、修改 kernel.git/driver/input/touchscreen/Kconfig, 在第十四行添加

```
config TOUCHSCREEN_S3C2410
    tristate "Samsung S3C2410 touchscreen input driver"
    depends on ARCH_S3C2410||ARCH_SMDK2410
    select SERIO
    help
config TOUCHSCREEN_S3C2410_DEBUG
    boolean "Samsung S3C2410 touchscreen debug messages"
    depends on TOUCHSCREEN_S3C2410
    help
```

4、修改配置

```
     Device Drivers  --->
         Input device support  --->
             [*]   Touchscreens  --->
             <*>   Samsung S3C2410 touchscreen input driver
```

5、修改 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

添加的部分用蓝色字体注出来。

。。。。。

```
    .lpcsel         = 0xf82,
```

```
};
//add     start
static struct s3c2410_ts_mach_info sbc2440_ts_cfg __initdata={
        .delay = 20000,
        .presc = 55,
        .oversampling_shift =2,
};
//end add
static struct platform_device *smdk2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c,
        &s3c_device_iis,
        &s3c_device_rtc, // add by panyingyun
        &s3c_device_ts, //add by panyingyun
};
static void __init smdk2440_map_io(void)
{
        s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
        s3c24xx_init_clocks(12000000);
        s3c24xx_init_uarts(smdk2440_uartcfgs, ARRAY_SIZE(smdk2440_uartcfgs));
        s3c24xx_init_touchscreen(&sbc2440_ts_cfg); // add by panyingyun
}
```

6、修改　kernel.git/include/asm-arm/plat-s3c24xx/devs.h

添加的部分用蓝色字体注出来

```
extern struct platform_device s3c_device_usbgadget;
//add by panyingyun
struct s3c2410_ts_mach_info{
        int    delay;
        int    presc;
        int    oversampling_shift;
};
void __init s3c24xx_init_touchscreen(struct s3c2410_ts_mach_info *hard_s3c2410_ts_info);
extern struct platform_device s3c_device_ts; //add by panyingyun
//end add
/* s3c2440 specific devices */
```

7、修改　kernel.git/arch/arm/ plat-s3c24xx/devs.c

添加的部分用蓝色字体注出来

```
struct platform_device s3c_device_adc = {
        .name               = "s3c2410-adc",
        .id             = -1,
        .num_resources          = ARRAY_SIZE(s3c_adc_resource),
        .resource       = s3c_adc_resource,
```

```
};

//add by panyingyun
/* Touchscreen*/
/* Serial port registrations */
/* Touchscreen */

static struct s3c2410_ts_mach_info s3c2410_ts_info;
/*
void __init set_s3c2410ts_info(struct s3c2410_ts_mach_info *hard_s3c2410ts_info)
{
	memcpy(&s3c2410ts_info,hard_s3c2410ts_info,sizeof(struct s3c2410_ts_mach_info));
	s3c_device_ts.dev.platform_data = &s3c2410ts_info;
}
EXPORT_SYMBOL(set_s3c2410ts_info);
*/
void __init s3c24xx_init_touchscreen(struct s3c2410_ts_mach_info *hard_s3c2410_ts_info)
{
memcpy(&s3c2410_ts_info, hard_s3c2410_ts_info, sizeof(struct s3c2410_ts_mach_info));
}
EXPORT_SYMBOL(s3c24xx_init_touchscreen);

struct platform_device s3c_device_ts = {
	.name		= "s3c2410-ts",
	.id		= -1,
	.num_resources	= ARRAY_SIZE(s3c_adc_resource),
	.resource	= s3c_adc_resource,
	.dev = {
		.platform_data = &s3c2410_ts_info,
	}
};

EXPORT_SYMBOL(s3c_device_ts);
//end add
```

注：在原先的文档中，还修改了 kernel.git/arch/arm/mach-s3c2410/clock.c.这里测试了一下，发现不修改，触摸屏依然可以工作，问题是点击的位置和实际的位置不一致，初步判断是没有校屏。

在 kernel.git/arch/arm/mach-s3c2410/clock.c 中使能 ads 时钟，即将下面代码由 init_clocks_disable 转移到 init_clocks 中。

```
{
	.name	=	“adc”,
	.id	=	-1,
	.parent	=	&clk_p,
	.enable	=	s3c2410_clkcon_enable,
```

```
    .ctrlbit    =    S3C2410_CLKCON_ADC,
},
```

注：参考网页：http://blog.163.com/yuan_xihua/blog/static/3074054420081183263029 3/

（16）USB Host 驱动移植

修改文件 kernel.git/arch/arm/mach-s3c2440/mach-s3c2440.c

```
//add by panyingyun    usb_host
#include <asm/arch/regs-clock.h>
#include <asm/arch/usb-control.h>
static struct s3c2410_hcd_info usb_info=
{
    .port[0]={.flags = S3C_HCDFLG_USED},
    .port[1]={.flags = S3C_HCDFLG_USED},
};
int usb_init(void)
{
    unsigned long upllvalue = (0x78<<12)|(0x02<<4)|(0x03);
    printk("USB Control, (c) 2008 panyingyun\n");
    s3c_device_usb.dev.platform_data = &usb_info;
    __raw_writel(upllvalue, S3C2410_UPLLCON);
    return 0;
}
//end add
static void __init smdk2440_map_io(void)
{
    s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
    s3c24xx_init_clocks(12000000);
    s3c24xx_init_uarts(smdk2440_uartcfgs, ARRAY_SIZE(smdk2440_uartcfgs));
    s3c24xx_init_touchscreen(&sbc2440_ts_cfg); // add by panyingyun
    usb_init(); // add by panyingyun usb host
}
```

测试：

添加 U 盘支持。这个只要在内核配置时，在 device drivers-->usb support 选项中，打开 USB Mass Storage support 支持。重新编者编译，即可支持 U 盘。但我移植的系统在插入 u 盘后出现

```
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices

scsi 0:0:0:0: Direct-Access     Generic  USB Flash Drive       PQ: 0 ANSI: 2
sd 0:0:0:0: [sda] 257152 512-byte hardware sectors (132 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 257152 512-byte hardware sectors (132 MB)
sd 0:0:0:0: [sda] Write Protect is off
```

```
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

此时，U 盘才起作用，用 mount -t vfat /dev/sda1 /mnt 命令挂载 u 盘后，可以正常读写了。

添加 usb 口鼠标，键盘的支持。选中内核 deveic drivers-->HID Device-->< > USB Human Interface Device (full HID) support，即可支持。测试是否正常工作的简单办法是插入鼠标，执行 cat /dev/mouse0 命令，此时，移动鼠标，会有乱码输出。表示鼠标工作正常。

（17）SD 卡驱动移植（参考已有的文档，没有成功，未测试）

将 driver_mmc_host 文件夹下的文件拷贝到 kernel.git/driver/mmc/host

具体修改可以参考源码，

替换 include 头文件，regs-sdi.h

参考：http://download.csdn.net/source/663488

扬创 utu2440 平台上 MMC Host 驱动程序(for linux2.6.26.5)。基于最新的内核做了一定修改！Have fun!

（18）uda1341 声卡驱动移植

拷贝 sound_oss 到目录 sound/oss 目录下。

配置选中 UDA1341 设备

1、将该文件放置到 sound/oss/目录下。 将 bitfield.h 放到 include/asm-arm/plat-s3c24xx/目录

2、在该目录下的 Makfile 文件的适当部位(和别的 obj 一起的地方)添加：

```
Obj-$(CONFIG_S3C2410_UDA1341)+= s3c_uda1341.o
```

以便能选择编译该文件。

3、在该目录下的 kconfig 文件的头部添加：

```
config S3C2410_UDA1341
      tristate "S3C2410 UDA1341 driver (S3C2440)"
      depends on SOUND_PRIME &&   SOUND && ARCH_S3C2410
      help
        The UDA1341 can be found in Samsung's S3C24XX
        platforms. If you have a board based on one
        of these. Say Y or M here.
        If unsure, say N.
```

以便能在 menuconfig 的时候能选择到这个声卡。

4、在 arch/arm/mach-s3c2440/mach-smdk2440.c 文件

这个可以不用，因为这里不是平台驱动的添加。添加会出现内核崩溃的现象。

```
static struct platform_device *smdk2440_devices[] __initdata = {
      ....
      ....
      //&s3c_device_iis,
      ....
      ....
};
```

注意：不用再添加地址映射！

5、make menuconfig 选择 driver->sound->oss->uda1341 ，选择对声卡的支持。编译完毕，下载到开发板，使用 madplay 播放 mp3 文件。效果不错。

参考：http://icode.csdn.net/source/663448

扬创 utu2440 板子上的声卡驱动(for linux2.6.26.5)

最新的 linux 内核在 dma 和 semaphore 结构上有一些变化，所以声卡驱动做了一些修改，保证能运行。Have fun!

2008.10.12 add: 将文件 s3c_uda1341.c 中的调用 access_ok()函数的地方都改为 !access_ok()。前几天测试移植好的 mplayer 时，没有声音。发现问题在这儿，修改了就好了。Linux 版本更新实在太快，一些基本的数据结构和宏改变都很大（boolean 型的返回值都能倒过来，sigh！），出的错误真令人哭笑不得。

2、Android 文件系统构建

主要参考

http://www.androidin.com/bbs/viewthread.php?tid=2741&extra=page%3D1&page=1

3、缺陷声明

1、内核部分

支持触摸屏（目前没有校屏程序，点击不准，需要修正）

支持 nfs 文件系统启动

支持 RTC(google 版经典时钟可以正确显示)

支持 DM9000 网卡

支持 yaffs 文件系统

电源管理

鼠标和键盘还存在问题（仅仅能识别，但不能正常使用）。

2、文件系统部分

clock /goole Browser/etc/ 等

第四章 基于 Android 平台的设备驱动开发及应用程序编写（未完成）

目标：在构建好的 Android 系统上做应用开发，包括驱动程序编写、java 程序编写、J2ME 程序移植。

1、mini2440 平台中断、串口、I/O 驱动编程

2、Android 的应用程序结构分析

3、Dalvik 和 Java VM 比较

4、J2ME 程序移植到 Android 平台的方法

5、基于 Dalvik 的 java 程序编写简述

参考论坛：

http://www.androidlab.cn/index.php
http://blog.chinaunix.net/u2/73521/article_0_1.html
http://www.yayabo.cn/forum-16-1.html
http://androidok.com/bbs/index.asp

参考网页：编译android原始码到模拟器上执行

http://www.dotblogs.com.tw/neil/archive/2009/04/03/7838.aspx

0.下载android的source code，请参考官方网页的做法吧！ （ http://source.android.com/download ）

1.取出编译的 kernel 设定档 ： (必须从模拟器那边去得到，因为要编译出的 image 要能在 emulator 上跑。)

1.1 打开模拟器： (你可以直接从 eclipse 那边开启或是直接在 cmd 下打 emulator -shell 来开启)

Ex:我的 sdk 放在 C:\WT\SoftWare\eclipse，那用 cmd 进入到 tools 目录下，

(1)视窗键+R，输入 cmd。

(2) cd C:\WT\SoftWare\eclipse\android-sdk-windows-1.1_r1\tools

(3) emulator -shell

1.2 拿出在模拟器中的设定档：

(1)再开启另一个 cmd，步骤如 1.1 的(1), (2)。

(2)输入 adb pull /proc/config.gz pulldata\config.gz (pull 是从模拟器中拿出档案,蓝色是来源,绿色的目的路径, config.gz 一般是放在 proc 目录下)

(3)解开 config.gz 档，得到 config，改变档名为.config。 (可以用 7-ZIP 来解或是使用 gunzip config.gz)

ps 如果你是在 windows 上的话，无法改成.config 的话，可以等丢到 kernel 目录中再改(mv config .config)。

1.3 拿到编译的设定档之后，需要把它放进 kernel 目录内：

(1)把.config放到~/mydroid/kernel下去覆盖原本的，如果kernel下原本就没有.config档时，可以输入make menuconfig ，再直接exit 。 (http://nmc.nchu.edu.tw/linux/kernel.htm)

2.修改 Makefile：

2.1 修改 android makefile：

(1)基本上，编译需要去设定编译的目的 os 和 cpu (target os and target cpu)，而 android 的 makefile 里的 os 和 cpu 预设是 linux 和 arm，也就是说编译后的程式能在 os 为 linux 且 cpu 为 arm 的机械中执行。所以如果丢到模拟器中跑的话，应该是不需要设定。

(2)如果需要设定可以去参考build/core/ envsetup. mk 这个档：

```
ifeq ($(TARGET_SIMULATOR),true)
ifneq ($(HOST_OS),linux)
$(error TARGET_SIMULATOR=true is only supported under Linux)
endif
TARGET_ARCH := $(HOST_ARCH)
TARGET_OS := $(HOST_OS)
else
ifeq ($(TARGET_ARCH),)
TARGET_ARCH := arm
endif
TARGET_OS := linux
endif
```

本身去执行编译动作的机械的os和cpu称为host os和host cpu，如果target与host不一致则需要cross compiler。 android预设的cross compiler位置可以从envsetup.mk档中可以找到，如下：

ABP:=$(ABP):$(PWD)/prebuilt/$(HOST_PREBUILT_TAG)/toolchain/arm-eabi-4.2.1/bin

2.2 修改 android kernel makefile：

(1)如果你不想直接修改 makefile，也可以在下 make 指令时一绪加入参数，如下。

make msm_defconfig ARCH=arm

make ARCH=arm CROSS_COMPILE = ../prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/bin/arm-eabi-

如果想直接去修改 makefile，需要设定 cross compile。 开启 kernel 目录下的 Makefile，并设定 CROSS_COMPILE 且注解掉 LDFLAGS_BUILD_ID，如下。

CROSS_COMPILE= **/android/arm-2008q3/bin/arm-none-linux-gnueabi-**

LDFLAGS_BUILD_ID = $(patsubst -Wl$(comma)%,%,$(call ld-option, -Wl$(comma)--build-id,))

corss compiler 可以使用 android source 里面原有的或是使用自己下载的来用。

原有的 corss compiler 的位置在 prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/bin/arm-eabi-

也可以从这边下载： http://www.codesourcery.com/sgpp/lite/arm/portal/release644

ps 其他修改设定可以去点阅参考网址。

3.编译 andoird 与 kernel：

3.1 编译 kernel：

(1)在编译的部份，要先编译 kernel 完后再编 android。 如果你没动到 kernel，则不需要再重新编译，可以去 prebuilt/android-arm/kernel/下找到 kernel-qemu 这个映像档来用。

(2) kernel 目录下的输入 make 开始编译。 (注意：需要完成步骤 1 的动作才会有.config，有设定档才能进行编译，要不然就要自己慢慢回答编译的选项！)

(3) *编译完，在 kernel/arch/arm/boot/目录下会产生 zImage。 此 zImage 就是新的 kernel image，原始的 image 是位于 sdk/tools/lib/images/kernel- qemu。

依android官方网页的emulator的说明(http://developer.android.com/guide/developing/tools/emulator.html)：

kernel-qemu.img ： The emulator-specific Linux kernel image.

3.2 编译 android：

(1)如果你先前就编译过 android(repo 时就 make 过)，可以输入 make clean 去清除旧的编译物。

(2)再来输入 make 就会看到一堆讯息啦！ (等它编译完吧！在我这边的 server 上是接近一个小时！)

(3)编译完，在./out/target/product/generic/下会发现有三个.img 档：system.img, ramdisk.img, userdata.img，依官方网页说明：

ramdisk.img ： The ramdisk image used to boot the system.

system.img ： The initial Android system image.

userdata.img : The initial user-data disk image

4.载入映像档到模拟器中：

4.1 映像档位置：

(1)依照 android 官方网页的说明，映像档是放在 lib/images 下，在我的电脑(os: xp)上是位于 C:\WT\Soft Ware\eclipse\ android-sdk-windows-1.1_r1\ tools\lib\images 。

用指令来指定 image 的方式如下：

emulator -image system.img -data userdata.img -ramdisk ramdisk.img

如果要重载 kernel 的 image 则如下：

emulator -kernel <your own path>/zImage

(2)如果直接把 image 档覆盖掉在 lib/images 的档案也能正常执行。

如果使用指令则会启动模拟器并载入参数路径所指的映像档，如下，其实上原本存在于 lib/images 的映像档并没有被参数载入的覆盖掉！

emulator -image backup\system.img

ps 有标注红星符号的地方 *表示这些步骤本人尚未进行测试，纯参考其他网页的做法。

参考网址：

http://www.cnblogs.com/fromsx/archive/2008/11/14/1333644.html

http://www.cnblogs.com/fromsx/archive/2008/11/14/1333693.html

http://www.androidin.com/learn/cn/200901/27-468.html

http://home.androidin.com/space.php?uid=18291&do=blog&id=27

http://blog.roodo.com/thinkingmore/archives/8533633.html

参考网页：http://www.mcuol.com/tech/116/29753.htm

（作者：北京理工大学 陈罡）google 的 android 很多人都希望在 gphone 没有出来之前，把它移植到相关的硬件平台上去。网上看了不少文章，总的感觉是：在这一步走得最远的就是 openmoko 的一个大师级别的黑客 Ben “Benno” Leslie,他曾经试图把目前 google 发布的 android 移植到 openmoko 的平台上去，并且做了 10000 多行代码的尝试。最终虽然由于 open moko 采用比较老的 arm 920t 的内核，而 android 采用较新的 arm926-ej-s 内核，而且使用了新的内核的一些新特性，导致移植失败，但是 anyway，他已经做了足够多的前期工作了，尔后的宣布成功移植 android 到 real target 板子上的人，大多是在他提供的 patch 的基础上继续走下去做出来的。

下面是一些有用的参考，希望有助于对此感兴趣的开发人员：
(1)Ben “Benno” Leslie 的关于 andorid 移植到 openmoko 的个人博客地址：
http://benno.id.au/blog/

(2)早期宣布成功移植 android 到 zauraus-sl-c760 的详细方法描述的链接：
http://euedge.com/blog/2007/12/06/google-android-runs-on-sharp-zaurus-sl-c760/

(3)后续的根据上述先行者们的工作，成功移植 android 到 zauraus-c3000 的方法：
http://androidzaurus.seesaa.net/article/74237419.html

(4)本文是参考下面的 wiki，接合个人的实践写出来的，对原文的作者表示一下感谢：
http://wiki.droiddocs.net/Compilation_of_Android_kernel

很羡慕这些人阿！

不过很可惜，偶的开发板是 s3c2410 的，恰好是 arm920t 的核心的。。。估计移植上去戏不是很大,需要重写很多代码，毕竟偶跟 benno 相差得太远太远了，同样是开发人员，差距咋就那么大呢？！
（毕竟 google 仅仅开放了 kernel 的源代码而已，他们需要开放的东西还很多。）

在这里把关于 android 内核编译方法简单写一下，或许对希望移植内核的朋友能有些帮助:
(看了 Benno 的移植过程以后，觉得即使你能够编译 google 开放出来的内核，意义也不是特别大，因为这个内核中加入了为了支持 qemu 的很多东西，而这些代码似乎对希望移植到真机上的朋友来说，没有任何意义，反而是一种阻碍)。

1)从 CodeSourcery 上面载用于交叉编译的工具链:
http://www.codesourcery.com/gnu_toolchains/arm/download.html
我在这里选择的是->ARM GNU/Linux，以及 IA32-GNU/Linux。有文章说应该选择 ARM EABI，我不知道了，没有测试过，反正我选择的这个编译的内核也是可以跑起来的:P

2)下载 google 的 android linux 的内核源代码：
http://code.google.com/p/android/downloads/list
主要是这个文件：linux-2.6.23-android-m3-rc20.tar.gz
3)把下载到的内核和交叉编译工具解压缩，并最好把工具链的路径放到 PATH 里面去
解压缩内核：

```
$ mkdir -p android
$ cd android
$ tar xzvf ../linux-2.6.23-android-m3-rc20.tar.gz
```

会解压出来一个叫做 kernel 的目录，google 的 android 的 linux 内核就在里面了。

解压缩交叉编译工具链：

```
$ cd /usr/local/
$ sudo cp ~/arm-2007q3-51-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 .
$ sudo tar zxvf arm-2007q3-51-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
```

此时会解压出来一个叫做 arm2007q3 的一个目录，这里面就是工具链了。

设置一下环境变量：

```
$ export PATH=$PATH:/usr/local/arm2007q3/bin
```

好了，到此，基本的内核编译环境就搞好了。

4)现在是要得到 android 的内核编译参数的配置文件的时候了,该文件需要从已经安装好的 android 的模拟器
中得到才行。所以安装 android 的 sdk 也是必须的，这一步不太明白的朋友可以参考我以前发的 android
命令行体验的文章。
首先启动 android 模拟器，然后通过 adb 得到模拟器中提供的内核配置文件：

```
$emulator &
$adb pull /proc/config.gz .
```

这时候 adb 工具会连接模拟器,并从它里面下载一个叫做 config.gz 的文件到你的当前目录下。
把它拷贝到你的 kernel 目录：

```
$cd ~/android/kernel
$cp ~/config.gz .
```

解压缩该文件，并重命名为.config，这一步做了就可以跳过 make menuconfig 之类的内核参数设置
动作了。

```
$gunzip config.gz
$mv config .config
```

5)修改 kernel 目录中的 Makefile 文件，用 emacs 或 vi 打开该 Makefile

修改 CROSS_COMPILE 变量为:

```
CROSS_COMPILE=arm-none-linux-gnueabi-
```

这个就是刚刚的下载和解压的工具链的前缀了，旨在告诉 make，在编译的时候要使用我们的工具链。

在 Makefile 中注释掉 LDFLAGS_BUILD_ID 这个变量：
例如将如下定义：

```
LDFLAGS_BUILD_ID = $(patsubst -Wl$(comma)%,%,\
                                $(call ld-option, -Wl$(comma)--build-id,))
```

修改为：

```
LDFLAGS_BUILD_ID=
#LDFLAGS_BUILD_ID = $(patsubst -Wl$(comma)%,%,\
#                               $(call ld-option, -Wl$(comma)--build-id,))
```

把它注释掉的原因是目前 android 的内核还不支持这个选项。--build-id 选项，主要是用于在生成的 elf
可执行文件中加入一个内置的 id，这样在 core dump，或者 debuginfo 的时候就可以很快定位这个模块是
哪次 build 的时候弄出来的。这样就可以避免，每次都把整个文件做一遍效验，然后才能得到该文件的是由
哪次 build 产生的。对于内核开发者来说，这是很不错的想法，可以节约定位模块版本和其影响的时间。

目前，该功能还出于 early stage 的状态，未来的 android 或许会支持，但至少目前的版本是不支持的。
所以，用#注释掉即可，或者害怕不保险的话，就加入 LDFLAGS_BUILD_ID=空，这样即使编译的时候用了，
也只是一个空格而已。
对这个--build-id 选项感兴趣的朋友，可以访问下面的网址，它的作者已经解释得非常明白了：
http://fedoraproject.org/wiki/Releases/FeatureBuildId

6)终于可以开始 make 了。
```
 $ make
```
不出意外的话，应该整个过程都会非常顺利，最终会在~/android/kernel/arch/arm/boot 目录下面
生成一个 zImage，这个就是我们要的内核映像了。

7)激动人心的时刻终于到来了，我们可以测试一下刚刚编译出来的内核可以不可以用了。
```
 $emulator -kernel ~/android/kernel/arch/arm/boot/zImage
```
当看到 red eye 在晃来晃去，最终显示出来 android 的界面的时候，一颗悬着的心总算放下了。

android 的 proc 里面的 version 如下：
```
# cat version
Linux version 2.6.23 (wayne@wayne) (gcc version 4.2.1 (CodeSourcery Sourcery G++ Lite 2007q3-51)) #1 Sat Jan 19 18:11:44 HKT 2008
```
从这里就可以看出，这是自己编译的 kernel，而不是人家 sdk 里面自带的 kernel-qemu 了。

android 自带的 sdk 里面的 kernel 映像的 version 应该是：
```
# cat version
Linux version 2.6.23-gcc3bc3b4 (arve@arvelnx.corp.google.com) (gcc version 4.2.1) #3 Tue Oct 30 16:28:18 PDT 2007
```
hoho, 这里不会把这个开发者的 email 暴露出来了吧。。。

android 的 cpuinfo 如下：
```
Processor       : ARM926EJ-S rev 5 (v5l)
BogoMIPS        : 313.75
Features        : swp half thumb fastmult vfp edsp java
CPU implementer : 0x41
CPU architecture: 5TEJ
CPU variant     : 0x0
CPU part        : 0x926
CPU revision    : 5
Cache type      : write-through
Cache clean     : not required
Cache lockdown  : not supported
```

```
Cache format    : Harvard
I size          : 4096
I assoc         : 4
I line length   : 32
I sets          : 32
D size          : 65536
D assoc         : 4
D line length   : 32
D sets          : 512

Hardware        : Goldfish
Revision        : 0000
Serial          : 0000000000000000
```

不过挺奇怪的，google sdk 自带的内核映像的 BogoMIPS 是 3.18 的，偶编译出来的是 3.13 的。

为了方便没有安装 sdk 的朋友也可以编译内核，我把 config.gz 贴上来：

http://www.androidin.com/bbs/viewthread.php?tid=2741&extra=page%3D1&page=1

参考网页：s3c2410/2440(armv4t) 移植教程及 android image 下载

相信国内很多朋友手上都有 s3c2410 / 2440 的片子，基于 armv4t(arm920t)的指令架构。在之前，因为 android 的一些底层代码含有 armv5t 的指令，所以以前无法移植到这样的平台。在这里也放出移植教程和已经编译好的 image。 让更多的朋友可以在自己的开发板上亲身体验 android。教程基于勤研 qt2410 以及扬创 utu2440 完成。

注意，移植是到 armv4 而不是 armv4t, 原因应该是不开 thumb 支持会比较好移植一些(改动未涉及的库依然会用 thumb 代码，所以 kernel 依然要开 thumb 支持)。

kernel 移植及 nfs 配置的详细部分等请自行查阅相关文档，本文只做提点，另外需要些 git 的操作。

目前的 image 只是临时方案。主要是基于 openmoko 的 benno 一些尚未正式发布的 patch，整体来说已经比较完善和 clean，我仅做了点小修改就能使用，由于部分库用 c 实现替换掉了 arm 实现，同时一些 critical 的 armv5 指令被 armv4 的替换，速度可能有点慢，尤其是 2410 上速度很慢。。对于严苛的 android 开发组来说，这些 patch 部分尚未提交，部分在 reviewing，还需要一些周期才会被接受进官方 git。等这些 patch 被官方接受后，我会整理另一套正式一些的版本，同时可能会提供些优化方案。

A. kernel 的移植

1. 移植，两个方案可以选。

1). 将 android 的官方 kernel 的补丁打到自己的 kernel 上，这个方法对于 kernel 版本比较新的朋友比较合适，也很简单。比如 2.6.24 或以上。

a. 下载 android kernel，执行 git diff v2.6.25 HEAD > android.patch

b. 进入你自己的 kernl，执行 git apply android.patch

c. 一般都会出 error，手动合并所有带 error 的文件，如果你是 2.6.25 的话很简单，2.6.24 会费点周折，因为 kobject/kset 的一些改动造成 android power / ipv4 等驱动不太好合。。

2). 将自己 kernel 上的 bsp 移植到 android 官方 kernel。适用于 kernel 比较旧的朋友。

因为 2410 / 2440 是 linux 原生支持，所以基本上只需要移植 lcd/touchpanel/网卡等等的驱动，并打开 2410 / 2440 支持选项即可。这个目前网上已经有大量的资料详细阐明，在此不再累述，但是出问题比较多的地方就在这里了。

2. 这些步骤完成后，检查如下选项是否都已经打开(make menuconfig)：

CONFIG_AEABI=y
CONFIG_SHMEM=y
CONFIG_ASHMEM=y
CONFIG_ARM_THUMB=y
CONFIG_ANDROID_BINDER_IPC=y
CONFIG_ANDROID_LOGGER=y
以下不一定重要， 但能开就开，除非移植有问题，比如 power 驱动：
CONFIG_ANDROID_POWER=y
CONFIG_ANDROID_POWER_STAT=y
CONFIG_ANDROID_POWER_ALARM=y
CONFIG_ANDROID_RAM_CONSOLE=y
CONFIG_ANDROID_TIMED_GPIO=y
CONFIG_ANDROID_PARANOID_NETWORK=y

3. 按需要 make zImage，再生成 uImage，烧写到 nand。

B. 制作 android nfs root

1. 下载我做好的 android_armv4 image 包，见帖子下方
2. 准备好能启动的 nfs root，设置好访问权限（重要!），资料请网上查阅，不再累述
3. 拷贝包中的 system 目录到 nfs_root/system, data 目录到 nfs_root/data, root 目录到 nfs_root/ （其他.img 文件可以模拟器使用，同时也可以做真实文件系统，后话了）
4. 设置 nfs 启动参数中的 init=/init，既指向 nfs_root 下的 init
5. 删除 nfs_root/init.rc，去掉除 mount tmpfs tmpfs /sqlite_stmt_journals size=4m 之外所有的 mount 命令

C. 运行!

跑跑看吧， 进入 android console 之后可以运行 logcat 查看 log。第一次启动会创建很多的数据，会比较慢，2410 几乎是折磨了。。平常的启动速度 2440 大概在 1 分多钟，2410 要 5 分钟了。看到漂亮的大手表，你就搞定了!!

常见问题：
1. lcd 不显示, 检查 framebuffer..如果正常，查看 nfs 权限。
2. 非法指令，检查 thumb，eabi 选项。
3. servicemanager 异常退出，检查 binder & ashmem 选项。
其他的看 kernel log 和 logcat 自己分析吧～ 呵呵～
20081208 修改：
最新进展, s3c2440 上也已经跑起来了,能进 idle, 触摸有 kernel 消息但是 android 还没认.
使用的是 utu2440f 平台, 扬创的..2410 太慢啦..上 2440 咯!
前段时间太忙没来得及更新和放 patch..
今天晚上直接放支持 2410/2440 的 android image..

原来的帖子(s3c2410):
目前已经能进入 idle..但屏幕颜色显示还有问题..触摸驱动也还没有搞..
硬件配置为 s3c2410 200mhz, 64m ram, 64m nand..
我的板子 kernel 比较老~直接用官方 android kernel 做的..于是所有驱动都得自己移植进来..后面估计有得搞了..
试了几个方案..感谢 openmoko 两位牛人..Sean 的 patch 没有跑起来..最终还是用的 Benno 的 patch..另外有些小修改..
启动速度一般, 感觉还行!
www.androidin.com/bbs/pub/armv4.tar.gz

参考网页：内核各个配置项及子项的含义

http://hi.baidu.com/shaotg/blog/item/1b0fd9ce612eba0293457eac.html
Linux 内核配置选项翻译 2.6.19.1
内容目录

说明 1：
文字中的 Y 表示选择进内核。M 表示编成模块。N 表示不选择。有的只能选 Y，有的只能选 M。
在 make menuconfig 下，＊表示 Y，M 表示 M，空白表示 N。
make xconfig 下，√ 表示 Y，· 表示 M，空白表示 N。

说明 2：
菜单中，有的选项你选了 Y 才出现，有的选了 Y，它反而不会出现。有的选项根本不会提供选择。所以我提供的菜单项应该不是最完整的设置选项。如果你碰到了新的选项，你可以认真看它的英文说明，或者通过上网搜索、在论坛提问等方式来了解它。
1. Code maturity level options

代码完成等级选项

1.1. Prompt for development and/or incomplete code/drivers

对开发中的或者未完成的代码和驱动进行提示。

LINUX 下的很多东西，比如网络设备、文件系统、网络协议等等，它们的功能、稳定性、或者测试等级等等还不能够符合大众化的要求，还处于开发之中。这就是所谓的阿尔法版本：最初开发版本；接下来的是 BETA 版本，公开测试版本。如果这是阿尔法版本，那么开发者为了避免收到诸如“为何这东西不工作”的信件的麻烦，常常不会让它发布出去。但是，积极的测试和使用阿尔法版本对软件的开发是非常好的。你只需要明白它未必工作得很好，在某些情况有可能会出问题。汇报详细的出错情况对开发者很有帮助。

这个选项同样会让一些老的驱动的可用。很多老驱动在将来的内核中已经被代替或者被移除。除非你想要帮助软件的测试，或者开发软件，或者你的机器需要这些特性，否则你可以选 N，那样你会在配置菜单中得到较少的选项。如果你选了 Y，你将会得到更多的阿尔法版本的驱动和代码的配置菜单。

2. General setup 常规安装选项

2.1. Local version - append to kernel release

在你的内核后面加上一串字符来表示版本。这些字符在你使用 uname -a 命令时会显示出来。你的字符最多不能超过 64 位。

2.2. Automatically append version information to the version string (LOCALVERSION_AUTO)

自动生成版本信息。这个选项会自动探测你的内核并且生成相应的版本，使之不会和原先的重复。这需要 Perl 的支持。

由于在编译的命令 make-kpkg 中我们会加入- - append-to-version 选项来生成自定义版本，所以这里选 N。

2.3. Support for paging of anonymous memory (swap)

这个选项将使你的内核支持虚拟内存，也就是让你的计算机好象拥有比实际内存更多的内存空间用来执行很大的程序。这个虚拟内存在 LINUX 中就是 SWAP 分区。除非你不想要 SWAP 分区，否则这里必选 Y。

2.4. System V IPC (IPC:Inter Process Communication)

中间过程连接是一组功能和系统调用，使得进程能够同步和交换信息。这通常来说是好事，有一些程序只有你选择了 Y 才能运行 。特别地，你想在 LINUX 下运行 DOS 仿真程序，你必须要选 Y。

你可以用 info ipc 命令来了解 IPC。

一定要选 Y。

2.4.1. IPC Namespaces (IPC_NS)

IPC 命名空间，命名空间的作用是区别同名的东西，就比如李宁和张宁，都叫“宁”，加个姓才能区分。这个选项也是为不同的服务器提供 IPC 的多命名，达到一个 IPC 提供多对象支持的目的。不清楚的话选 N。

2.5. POSIX Message Queues 可移植操作系统接口信息队列

可移植操作系统接口信息队列是 IPC 的一部分，在通信队列中有较高的优先权来保持通信畅通。如果你想要编译和运行在 Solaris 操作系统上写的 POSIX 信息队列程序，选 Y，同时你还需要 mqueue 库来支持这些特性。它是作为一个文件系统存在（mqueue），你可以 mount 它。为保证以后的不同程序的协同稳定，如果不清楚，选 Y。

2.6. BSD Process Accounting BSD 进程统计

如果你选 Y，用户级别的程序就可以通过特殊的系统调用方式来通知内核把进程统计信息记录到一个文件，当这个进程存在的时候，信息就会被内核记录进文件。信息通常包括建立时

间、所有者、命令名称、内存使用、控制终端等。这对用户级程序非常有用。所以通常选 Y 是一个好主意。

2.6.1. BSD Process Accounting version 3 file format

选 Y，统计信息将会以新的格式（V3）写入，这格式包含进程 ID 和父进程。注意这个格式和以前的 v0/v1/v2 格式不兼容，所以你需要升级相关工具来使用它。

2.7. Export task/process statistics through netlink (EXPERIMENTAL)

处于实验阶段的功能。通过通用的网络输出工作/进程的相应数据，和 BSD 不同的是，这些数据在进程运行的时候就可以通过相关命令访问。和 BSD 类似，数据将在进程结束时送入用户空间。如果不清楚，选 N。

2.8. UTS Namespaces

通用终端系统的命名空间。它允许容器，比如 Vservers 利用 UTS 命名空间来为不同的服务器提供不同的 UTS。如果不清楚，选 N。

2.9. Auditing support 审计支持(AUDIT)

允许审计的下层能够被其他内核子系统使用，比如 SE－Linux，它需要这个来进行登录时的声音和视频输出。没有 CONFIG_AUDITSYSCALL 时（即下一个选项）无法进行系统调用。

2.9.1. Enable system-call auditing support (AUDITSYSCALL)

允许系统独立地或者通过其他内核的子系统，调用审计支持，比如 SE－Linux。要使用这种审计的文件系统来查看特性，请确保 INOTIFY 已经被设置。

上一项的子选项，两项要选就都选。我并不清楚审计的意义，可能是为了调用其他内核的东西吧。所以就都选了，因为我机器上还有个官方 2.6.15-27 内核。

2.10. Kernel .config support

这个选项允许.config 文件（即编译 LINUX 时的配置文件）保存在内核当中。

它提供正在运行中的或者还在硬盘中的内核的相关配置选项。可以通过内核镜像文件 kernel image file 用命令 script scripts/extract-ikconfig 来提取出来，作为当前内核重编译或者另一个内核编译的参考。如果你的内核在运行中，可以通过/proc/config.gz 文件来读取。下一个选项提供这项支持。

看起来好像是一个不错的功能，可以把编译时的 .config 文件保存在内核中，以供今后参考调用。用来重编译和编译其他的内核的时候可以用上。你是一个编译内核的狂人的话，这项要选上（比如我，不过我总是会备份所有的 .config 文件）。

2.10.1. Enable access to .config through /proc/config.gz

上一项的子项，可以通过/proc/config.gz 访问当前内核的.config 。新功能，上一项选的话这个就选上吧。

2.11. Cpuset support

多 CPU 支持。这个选项可以让你建立和管理 CPU 集群，它可以动态地将系统分割在各个 CPU 和内存节点中，在各个节点是独立运行的。这对大型的系统尤其有效。

如果不清楚，选 N。

2.12. Kernel->user space relay support (formerly relayfs)

内核系统区和用户区进行传递通讯的支持。这个选项在特定的文件系统中提供数据传递接口支持，它可以提供从内核空间到用户空间的大批量的数据传递工具和设施。

如果不清楚，选 N。

2.13. Initramfs source file(s)

没有可选项。

2.14. Optimize for size (Look out for broken compilers!)

这个选项将在 GCC 命令后用 "-Os" 代替 "-O2"参数，这样可以得到更小的内核。警告：某

些 GCC 版本会导致错误。如果有错，请升级你的 GCC。
如果不清楚，选 N。
这是优化内核大小的功能，没必要选。一个编译好的内核才 7－10 多 M，大家不会少这么点空间吧。选上了可能会出一些问题。最好不选。
2.15. Configure standard kernel features (for small systems)
这个选项可以让内核的基本选项和设置无效或者扭曲。这是用于特定环境中的，它允许“非标准”内核。你要是选它，你一定要明白自己在干什么。
这是为了编译某些特殊用途的内核使用的，例如引导盘系统。通常你可以不选择这一选项，你也不用关心他的子选项。
3. Loadable module support 引导模块支持
3.1. Enable loadable module support
这个选项可以让你的内核支持模块，模块是什么呢？模块是一小段代码，编译后可在系统内核运行时动态的加入内核，从而为内核增加一些特性或是对某种硬件进行支持。一般一些不常用到的驱动或特性可以编译为模块以减少内核的体积。在运行时可以使用 modprobe 命令来加载它到内核中去(在不需要时还可以移除它)。一些特性是否编译为模块的原则是，不常使用的，特别是在系统启动时不需要的驱动可以将其编译为模块，如果是一些在系统启动时就要用到的驱动比如说文件系统，系统总线的支持就不要编为模块，否则无法启动系统。在启动时不用到的功能，编成模块是最有效的方式。你可以查看 MAN 手册来了解：modprobe, lsmod, modinfo, insmod 和 rmmod.
如果你选了这项，你可能需要运行 "make modules_install" 命令来把模块添加到/lib/modules/目录下，以便 modprobe 可以找到它们。
如果不清楚，选 Y。
3.2. Module unloading
这个选项可以让你卸载不再使用的模块，如果不选的话你将不能卸载任何模块(有些模块一旦加载就不能卸载，不管是否选择了这个选项)。
如果不清楚，选 Y。
3.2.1. Forced module unloading
这个选项允许你强行卸除模块，即使内核认为这不安全。内核将会立即移除模块，而不管是否有人在使用它（用 rmmod -f 命令）。这主要是针对开发者和冲动的用户提供的功能。
如果不清楚，选 N。
3.3. Module versioning support (MODVERSIONS)
有时候，你需要编译模块。有时候，你需要编译模块。选这项会添加一些版本信息，来给编译的模块提供独立的特性，以使不同的内核在使用同一模块时区别于它原有的模块。这有时可能会有点用。
如果不清楚，选 N。
3.4. Source checksum for all modules
这个功能是为了防止你在编译模块时不小心更改了内核模块的源代码但忘记更改版本号而造成版本冲突。
如果不清楚，选 N。
3.5. Automatic kernel module loading
允许内核自动加载模块。一般情况下，如果我们的内核在某些任务中要使用一些被编译为模块的驱动或特性时，我们要先使用 modprobe 命令来加载它，内核才能使用。不过，如果你选择了这个选项,在内核需要一些模块时它可以自动调用 modprobe 命令来加载需要的模块。
如果不清楚，选 Y。

4. Block layer
块设备。
4.1. Enable the block layer (BLOCK)
这选项使得块设备可以从内核移除。如果不选，那么 blockdev 文件将不可用，一些文件系统比如 ext3 将不可用。这个选项会禁止 SCSI 字符设备和 USB 储存设备，如果它们使用不同的块设备。
选 Y，除非你知道你不需要挂载硬盘和其他类似的设备。不过此项无可选项。
4.1.1. Support for Large Block Devices (LBD)
如果你要用大于 2TB 的硬盘，选这个。
4.1.2. Support for tracing block io actions
对块设备进行跟踪和分析的功能。
4.1.3. Support for Large Single Files (LSF)
大文件支持。如果你准备建的文件大于 2TB，选这个。
4.1.4. IO Schedulers 磁盘 I / O 调度器
I / O 是输入输出带宽控制，主要针对硬盘，是核心的必须的东西。这里提供了三个 IO 调度器。
4.1.4.1. Anticipatory I/O scheduler
抢先式 I/O 调度方式是默认的磁盘调度方式。它对于大多数环境通常是比较好的选择。但是它和 Deadline I/O 调度器相比有点大和复杂，它有时在数据调入时会比较慢。
4.1.4.2. Deadline I/O scheduler
Deadline I / O 调度器简单而又紧密，在性能上和抢先式调度器不相上下，在一些数据调入时工作得更好。至于在单进程 I / O 磁盘调度上，它的工作方式几乎和抢先式调度器相同，因此也是一个好的选择。
看介绍这个好像比上面的更好，可以试试。不过按照我的平衡观点，好东西都会带来问题。
4.1.4.3. CFQ I/O scheduler
CFQ 调度器尝试为所有进程提供相同的带宽。它将提供平等的工作环境，对于桌面系统很合适。
4.1.4.4. Default I/O scheduler 选择默认的 I / O 调度器
我选了 Anticipatory I/O scheduler。
我这样理解上面三个 IO 调度器：抢先式是传统的，它的原理是一有响应，就优先考虑调度。如果你的硬盘此时在运行一项工作，它也会暂停下来先响应用户。
期限式则是：所有的工作都有最终期限，在这之前必须完成。当用户有响应时，它会根据自己的工作能否完成，来决定是否响应用户。
CFQ 则是平均分配资源，不管你的响应多急，也不管它的工作量是多少，它都是平均分配，一视同仁的。
5. Processor type and features 处理器类型及特性
5.1. Symmetric multi-processing support (SMP) 对称多处理器支持。
这将支持有多 CPU 的系统。如果你的系统只有一个 CPU，选 N。反之，选 Y。
如果你选 N，内核将会在单个或者多个 CPU 的机器上运行，但是只会使用一个 CPU。如果你选 Y，内核可以在很多（但不是所有）单 CPU 的机器上运行，在这样的机器，你选 N 会使内核运行得更快。
注意如果你选 Y，然后在 Processor family 选项中选择"586" or "Pentium" ，内核将不能运行在 486 构架的机器上。同样的，多 CPU 的运行于 PPro 构架上的内核也无法在 Pentium 系列的板上运行。

使用多 CPU 机器的人在这里选 Y，通常也会在后面的选项“Enhanced Real Time Clock Support”中选 Y。如果你在这选 Y，"Advanced Power Management" 的代码将不可用。
如果不清楚，选 N。
5.2. Subarchitecture Type 子构架类型
5.2.1. PC-compatible
选这个如果你的机器是标准 PC
5.2.2. AMD Elan
注意，如果你是 K6/Athlon/Opteron 处理器不要选这个
5.2.3. Voyager
5.2.4. NUMAQ (IBM/Sequent)
5.2.5. Summit/EXA (IBM x440)
5.2.6. SGI 320/540 (Visual Workstation)
5.2.7. Generic architecture (Summit, bigsmp, ES7000, default)
5.2.8. Support for Unisys ES7000 IA32 series
5.3. Processor family
处理器类型。针对自己的 CPU 类型，选取相应的选项。
这里是处理器的类型。这里的信息主要目的是用来优化。为了让内核能够在所有 X86 构架的 CPU 上运行（虽然不是最佳速度），在这你可以选 386。
内核不会运行在比你选的构架还要老的机器上。比如，你选了 Pentium 构架来优化内核，它将不能在 486 构架上运行。
如果你不清楚，选 386。
5.3.1. - "386"
5.3.2. - "486"
5.3.3. - "586"
5.3.4. - "Pentium-Classic"
5.3.5. - "Pentium-MMX"
5.3.6. - "Pentium-Pro"
5.3.7. - "Pentium-II"
5.3.8. - "Pentium-III"
5.3.9. - "Pentium-4"
5.3.10. - K6, K6-II and K6-III
5.3.11. - "Athlon" K7 (Athlon/Duron/Thunderbird).
5.3.12. -Opteron/Athlon64/Hammer/K8
5.3.13. - "Crusoe"
5.3.14. - "Efficeon"
5.3.15. - "Winchip-C6"
5.3.16. - "Winchip-2"
5.3.17. - "Winchip-2A"
5.3.18. - "GeodeGX1"
5.3.19. - "Geode GX/LX"
5.3.20. - "CyrixIII/VIA C3"
5.3.21. - VIA C3-2 "Nehemiah".
5.4. Generic x86 support
通用 X86 支持。

除了对上面你选择的 X86 CPU 进行优化，它还对更多类型 X86 CPU 的进行优化。这将会使内核在其他的 X86 CPU 上运行得更好。
对于供应商来说，他们非常需要这些功能，因为他们需要更通用的优化支持。
这个选项提供了对 X86 系列 CPU 最大的兼容性，用来支持一些少见的 x86 构架的 CPU。如果你的 CPU 能够在上面的列表中找到，就里就不用选了。
5.5. HPET Timer Support
HPET 时钟支持
允许内核使用 HPET 。HPET 是代替当前 8254 的下一代时钟，全称叫作高精度事件定时器。你可以安全地选 Y。但是，HEPT 只会在支持它的平台和 BIOS 上运行。如果不支持，8254 将会激活。
选 N，将继续使用 8254 时钟。
5.6. Maximum number of CPUs (2-255)
设置最高支持的 CPU 数，无法选择。我的显示为 8。
5.7. SMT (Hyperthreading) scheduler support
超线程调度器支持
超线程调度器在某些情况下将会对 Intel Pentium 4 HT 系列有较好的支持。
如果你不清楚，选 N。
5.8. Multi-core scheduler support
多核调度机制支持，双核的 CPU 要选。
多核心调度在某些情况下将会对多核的 CPU 系列有较好的支持。
如果你不清楚，选 N。
5.9. Preemptible Kernel 抢先式内核。
一些优先级很高的程序可以先让一些低优先级的程序执行，即使这些程序是在核心态下执行。从而减少内核潜伏期，提高系统的响应。当然在一些特殊的点的内核是不可抢先的，比如内核中的调度程序自身在执行时就是不可被抢先的。这个特性可以提高桌面系统、实时系统的性能。
下面有三个选项：
5.9.1. No Forced Preemption (Server) 非强迫式抢先。
这是传统的 LINUX 抢先式模型，针对于高吞吐量设计。它同样在很多时候会提供很好的响应，但是也可能会有较长的延迟。
如果你是建立服务器或者用于科学运算，选这项，或者你想要最大化内核的原始运算能力，而不理会调度上的延迟。
5.9.2. Voluntary Kernel Preemption (Desktop) 自动式内核抢先
这个选项通过向内核添加更多的“清晰抢先点”来减少内核延迟。这些新的抢先点以降低吞吐量的代价，来降低内核的最大延迟，提供更快的应用程序响应。这通过 允许低优先级的进程自动抢先来响应事件，即使进程在内核中进行系统调用。这使得应用程序运行得更“流畅”，即使系统已经是高负荷运转。
如果你是为桌面系统编译内核，选这项。
5.9.3. Preemptible Kernel (Low-Latency Desktop) 可抢先式内核（低延迟桌面）
这个选项通过使所有内核代码（非致命部分）编译为“可抢先”来降低内核延迟。
这通过允许低优先级进程进行强制抢先来响应事件，即使这些进程正在进行系统调用或者未达到正常的“抢先点”。这使得应用程序运行得更加“流畅”即使系统已经是高负荷运转。代价是吞吐量降低，内核运行开销增大。
选这项如果你是为桌面或者嵌入式系统编译内核，需要非常低的延迟。

如果你要最快的响应，选第三项。我认为万物是平衡的，低延迟意味着系统运行不稳定，因为过多来响应用户的要求，所以我选第二个。

5.10. Preempt The Big Kernel Lock

抢先式大内核锁(早期 Linux 用于支持 SMP 系统时所采用的非细粒度锁)

这个选项通过让大内核锁变成“可抢先”来降低延迟。

选 Y 如果你在构建桌面系统。如果你不清楚，选 N。

5.11. Machine Check Exception 机器例外检查

机器例外检查允许处理器在检测到问题（比如过热、组件错误）时通知内核。内核根据问题的严重程度来决定下一步行为，比如在命令行上打印告警信息，或者关 机。你的处理器必须是 Pentium 或者更新版本才能支持这个功能。 用 cat /proc/cpuinfo 来检测你的 CPU 是否有 mce 标志。

注意一些老的 Pentium 系统存在设计缺陷，会提供假的 MCE 事件，所以在所有 P5 处理器上 MCE 被禁用，除非在启动选项上明确 "mce"参数。同样地，如果 MCE 被编译入内核并在非标准的机器上导致错误，你可以用"nomce"启动参数来禁用 MCE。

MCE 功能会自动忽视非 MCE 处理器，比如 386 和 486，所以几乎所有人都可以在这里选 Y。

5.11.1. Check for non-fatal errors on AMD Athlon/Duron / Intel Pentium 4

检测 AMD Athlon/Duron / Intel Pentium 4 的非致命错误

允许这项特性，系统将会启动一个计时器，每 5 秒进行检测。非致命问题会自动修正（但仍然会记录下来），如果你不想看到这些信息，选 N。这些信息可以让你发现要损坏的硬件，或者是非标准规格硬件（比如：超频的）。

这个功能只会在特定的 CPU 上起作用。

5.11.2. check for P4 thermal throttling interrupt.

检测 P4 节能器中断

当 P4 进入节能状态时，打印信息。

5.12. Toshiba Laptop support 东芝笔记本支持。

这个选项是针对 Toshiba 笔记本的，可以用来访问 Toshiba 的系统管理模式，可以直接设置 BIOS。不过要注意它只在 Toshiba 自己的 BIOS 中起作用。假如你有一台 Toshiba 笔记本，而它的 BIOS 是 Phoenix 的，那这个选项仍然是无用的。

5.13. Dell laptop support

DELL 笔记本支持。功能同上

5.14. Enable X86 board specific fixups for reboot

X86 板的重启修复功能。

这将打开芯片或者主板上的重启修复功能 ，从而能够使之正常工作。这功能仅仅在一些硬件和 BIOS 的特定组合上需要。需要这项功能的征兆是重启时使系统卡死或者挂起。

目前，这个修复功能仅仅支持 Geode GX1/CS5530A/TROM2.1.的组合。

选 Y 如果你需要这项功能，目前，选 Y 是安全的，即使你不需要它。否则，选 N。

5.15. /dev/cpu/microcode - Intel IA32 CPU microcode support

是否支持 Intel IA32 架构的 CPU。这个选项将让你可以更新 Intel IA32 系列处理器的微代码，显然你需要到网上去下载最新的代码，LINUX 不提供这些代码。当然你还必须在文件系统选项中选择/dev file system support 才能正常的使用它。如果你把它译为模块 ,它将是 microcode。

IA32 主要用于高于 4GB 的内存。详见下面的“高内存选项”。

5.16. /dev/cpu/*/msr - Model-specific register support

是否打开 CPU 特殊功能寄存器的功能。这个选项桌面用户一般用不到，它主要用在 Intel 的

嵌入式 CPU 中的，这个寄存器的作用也依赖与不同的 CPU 类型而有所不同，一般可以用来改变一些 CPU 原有物理结构的用途，但不同的 CPU 用途差别也很大。

5.17. /dev/cpu/*/cpuid - CPU information support

是否打开记录 CPU 相关信息功能。这会在/dev/cpu 中建立一系列的设备文件，用以让过程去访问指定的 CPU。

5.18. High Memory Support (4GB) 高容量内存支持

LINUX 能够在 X86 系统中使用 64GB 的物理内存。但是，32 位地址的 X86 处理器只能支持到 4GB 大小的内存。这意味着，如果你有大于 4GB 的物理内存，并非都能被内核“永久映射”。这些非永久映射内存就称为“高阶内存”。

如果你编译的内核永远都不会运行在高于 1G 内存的机器上，选 OFF（默认选项，适合大多数人）。这将会产生一个"3GB/1GB"的内存空间划分，3GB 虚拟内存被内核映射以便每个处理器能够“看到”3GB 的虚拟内存空间，这样仍然能够保持 4GB 的虚拟内存空间被内核使用，更多的物理内存能够被永久映射。

如果你有 1GB－4GB 之间的物理内存，选 4GB 选项。如果超过 4GB，那么选择 64GB。这将打开 Intel 的物理地址延伸模式（PAE）。PAE 将在 IA32 处理器上执行 3 个层次的内存页面。PAE 是被 LINUX 完全支持的，现在的 Intel 处理器 (Pentium Pro 和更高级的)都能运行 PAE 模式。注意：如果你选 64GB，那么在不支持 PAE 的 CPU 上内核将无法启动。

你机器上的内存能够被自动探测到，或者你可以用类似于"mem=256M"的参数强制给内核指定内存大小。

5.18.1. off 如果不清楚，选 OFF。

5.18.2. 4GB 选这项如果你用的是 32 位的处理器，内存在 1-4GB 之间。

5.18.3. 64GB 选这项如果你用的是 32 位的处理器，内存大于 4GB。

5.19. Memory model 内存模式

5.19.1. Flat Memory 平坦内存模式。

这个选项允许你改变内核在内部管理内存的一些方式。大多数用户在这只会有一个选项：Flat Memory。这是普遍的和正确的选项。

一些用户的机器有更高级的特性，比如 NUMA 和内存热拔插，那将会有不同的选项。Discontiguous Memory（非接触式内存模式）是一个更成熟、更好的测试系统。但是对于内存热拔插系统不太合适，会被"Sparse Memory"代替。如果你不清楚"Sparse Memory"和"Discontiguous Memory"的区别，选后者。

如果不清楚，就选 Flat Memory。

5.19.2. Sparse Memory 稀疏内存模式。

这对某些系统是唯一选项，包括内存热拔插系统。这正常。

对于其他系统，这将会被 Discontiguous Memory 选项代替。这个选项提供潜在的更好的特性，可以降低代码复杂度，但是它是新的模式，需要更多的测试。

如果不清楚，选择"Discontiguous Memory" 或 "Flat Memory"。

我的机器上只有这两个选项，我选 Flat Memory。

5.20. 64 bit Memory and IO resources (EXPERIMENTAL)

64 位内存和 IO 资源

这个选项将使内存和 IO 资源变成 64 位的。

实验选项，可以让内存和 I / O 变为 64 位。我的总线是 32 位的，所以还是不选了。选了不知道会不会出错。

5.21. Math emulation

数学仿真

LINUX 可以仿真一个数学协处理器（用来进行浮点运算），如果你没有的话。486DX 和 Pentium 处理器内建有数学协处理器。486SX 和 386 的没有，除非你专门加过 487DX 或者 387 协处理器。所有人都需要协处理器或者这个仿真。
如果你没有数学协处理器，你需要在这选 Y。如果你有了协处理器还在这选 Y，你的协处理器仍然被用到。这意味着如果你打算把编译的内核用在不同的机器上，选 Y 是明智的选择。如果不清楚，选 Y，这将使内核增加 66KB，无伤大雅。

5.22. MTRR (Memory Type Range Register) support
内存类型区域寄存器
在 Intel P6 系列处理器(Pentium Pro, Pentium II 和更新的)上，MTRR 将会用来规定和控制处理器访问某段内存区域的策略。
如果你在 PCI 或者 AGP 总线上有 VGA 卡，这将非常有用。例如可将 MTTR 设为在显存的地址范围上使用“write-combining”策略，这样 CPU 可以在 PCI/AGP 总线爆裂之前将多次数据传输集合成一个大的数据传输，这样可以提升图像的传送速度 2.5 倍以上。选 Y，会生成文件 /proc/mtrr，它可以用来操纵你的处理器的 MTRR。典型地，X server 会用到。
这段代码有着通用的接口，其他 CPU 的寄存器同样能够使用该功能。Cyrix 6x86, 6x86MX 和 M II 处理器有 ARR ，它和 MTRR 有着类似的功能。AMD K6-2/ K6-3 有两个 MTRR，Centaur C6 有 8 个 MCR 允许复合写入。所有这些处理器都支持这段代码，你可以选 Y 如果你有以上处理器。
选 Y 同样可以修正 SMP BIOS 的问题，它仅为第一个 CPU 提供 MTRR，而不为其他的提供。这会导致各种各样的问题，所以选 Y 是明智的。
你可以安全地选 Y，即使你的机器没有 MTRR。这会给内核增加 9KB。

5.23. Boot from EFI support
EFI 启动支持
这里允许内核在 EFI 平台上使用储存于 EFI 固件中的系统设置启动。这也允许内核在运行时使用 EFI 的相关服务。
这个选项只在有 EFI 固件的系统上有用，它会使内核增加 8KB。另外，你必须使用最新的 ELILO 登录器才能使内核采用 EFI 的固件设置来启动（GRUB 和 LILO 完全不知道 EFI 是什么东西）。即使你没有 EFI，却选了这个选项，内核同样可以启动。
大家应该用的是 GRUB，所以选上这个也没什么用。

5.24. Enable kernel irq balancing (IRQBALANCE) 中断平衡。
这个选项使系统进行中断平衡。
如果你是双核 CPU，如果不选这项，那么中断负荷都在第一个 CPU 上，其他的 CPU 可能得不到中断。

5.25. Use register arguments (REGPARM) 寄存器参数使用。
使用寄存器参数
用‘-mregparm=3’的参数编译内核。这使 gcc 使用更高效的应用程序二进制接口（ABI）来跳过编译时的前三个调用寄存器参数。这使得代码编译更精巧更快速。
如果你不选这个选项，默认的 ABI 将会使用。
如果不清楚，选 Y。

5.26. Enable seccomp to safely compute untrusted bytecode (SECCOMP)
允许 SECCOMP（快速计算）安全地运算非信任代码。
这个内核特性在程序出现数码错误，需要重新对非信任的代码进行运算时非常有效。它使用管道或者其他传输方式，使文件描述进程支持读/写的系统调用，这样可以利用 SECCOMP 隔离那些程序本身的空间。

一旦 seccomp 通过/proc/<pid>/seccomp 运行，它将不能停止，任务也只能进行一些安全的被 seccomp 认证的系统调用。
如果不清楚，选 Y。只有嵌入式系统选 N。
5.27. Timer frequency 时钟频率
允许设置时钟频率。
这是用户定义的时钟中断频率 100HZ-1000 HZ ，不过 100 HZ 对服务器和 NUMA 系统更合适，它们不需要很快速的响应用户的要求，因为时钟中断会导致总线争用和缓冲打回。注意在 SMP 环境中，时钟中断由变量 NR_CPUS * Hz 定义在每个 CPU 产生。
其实和前面的抢先式进程差不多，就是多少频率来响应用户要求。我选了 250HZ 的。要快点的可以选 1000HZ 的。但是还是那句话，一切是平衡的。机器过快响应你，它自己的活就不知道做得好不好了。
5.27.1. 100 HZ (HZ_100)
100 HZ 是传统的对服务器、SMP 和 NUMA 的系统选项。这些系统有比较多的处理器，可以在中断较集中的时候分担中断
5.27.2. 250 HZ (HZ_250)
250 HZ 对服务器是一个好的折衷的选项，它同样在 SMP 和 NUMA 系统上体现出良好的反应速度。
5.27.3. 1000 HZ (HZ_1000)
1000 HZ 对于桌面和其他需要快速事件反应的系统是非常棒的。
5.28. kexec system call (KEXEC) 快速重启调用。
kexec 系统调用
kexec 是一个用来关闭你当前内核，然后开启另一个内核的系统调用。它和重启很像，但是它不访问系统固件。由于和重启很像，你可以启动任何内核，不仅仅是 LINUX。
kexec 这个名字是从 exec 系统调用来的。它只是一个进程，可以确定硬件是否正确关闭，所以如果这段代码没能正确为你进行初始化工作，请不要奇怪。它对设备的热拔插会有点帮助。由于它对硬件接口会乱写点东西，所以我没什么好的建议给你。
Linus 本人都没话说，估计是受害不浅。我们当然不能上当，选 N！
5.29. Support for hot-pluggable CPUs (EXPERIMENTAL)
对热拔插 CPU 的支持
选 Y，可以做个实验，把 CPU 关闭和打开，也可以中止 SMP 系统。CPU 可以通过/sys/devices/system/cpu 来进行控制。
5.30. Compat VDSO support (COMPAT_VDSO)
Compat VDSO 支持
如果你运行的是最新的 glibc（GNU C 函数库）版本（ 2.3.3 或更新），选 N，这样可以移除高阶的 VDSO 映射，使用随机的 VDSO。
如果不清楚，选 Y。
5.31. Firmware Drivers 固件驱动。
固件就是你板上的 BIOS、各种显卡芯片之类的已经固化好的记录某些特定数据的东西。
5.31.1. BIOS Enhanced Disk Drive calls determine boot disk
BIOS 加强磁盘功能，确定启动盘。
选 y 或 M，如果你要使用 BIOS 加强磁盘服务功能来确定 BIOS 用哪个磁盘来启动。启动后这个信息会反映在系统文件中。
这个选项是实验性的，而且已经被确认在某些未测试选项下会启动失败。很多磁盘控制器的 BIOS 供应商都不支持这个特性。

5.31.2. BIOS update support for DELL systems via sysfs
用于 DELL 机器的 BIOS 升级支持。
5.31.3. Dell Systems Management Base Driver (DCDBAS)
DELL 系统管理器的基本驱动。
6. Power management options (ACPI, APM)
电源管理选项（ACPI、APM）
6.1. Power Management support
电源管理支持
电源管理意味着你电脑上的某一部分在不用的时候可以关闭或者休眠。这领域有两个竞争对手：APM 和 ACPI。如果你需要两者之一，请把这里选上，再把下面的相关内容选上。
电源管理对于使用电池的笔记本相当重要。如果你有笔记本，请参照几个网站上的说明。
注意，即使你在这选 N，在 X86 构架的机器上，LINUX 会发出 hlt 指令如果没有任务，因此会让处理器休眠，达到节电的目的。
6.1.1. Legacy Power Management API (PM_LEGACY)
电源管理继承接口
为 pm_register() （电源管理寄存器）和同类寄存器提供支持。
如果不清楚，选 Y。
6.1.2. Power Management Debug Support
电源管理调试支持
这个选项提供详细的电源管理调试信息。当你调试和报告电源管理漏洞的时候非常有用，有点像电源管理的“中断”支持。
6.1.3. Driver model /sys/devices/.../power/state files (DEPRECATED)
驱动模式文件 /sys/devices/.../power/state （不赞成使用）
这个驱动模式通过系统文件类型启动，试图来给电源管理设备提供用户空间连通装置。这个特性从来没有能很好地工作过，除非是用来进行测试，否则它处在被移除之列。我们不清楚用通用的方法能否进行各种各样的设备电源管理，目前是专用的总线和驱动来替代相关功能。
6.2. ACPI Support 高级电源配置接口支持
高级电源设置接口（ACPI）支持需要整合了 ACPI 的平台（固件/硬件），并且这个平台要支持操作系统和电源管理软件的设置。这个选项会给你的内核增加 70KB。
LINUX ACPI 提供了相当强大的电源接口，甚至可以取代一些传统的设置和电源管理接口，包括 PNP BIOS（即插即用 BIOS）规范，MPS（多处理器规范），和 APM（高级电源管理）规范。如果 ACPI 和 APM 同时被选上，先被系统调用的起作用。
6.2.1. AC Adapter AC 交流电源适配器
这个驱动给 AC 交流电源适配器提供支持，它指示出系统是否在 AC 下工作。如果你的系统可以在 AC 和电池状态下切换，选 Y。
6.2.2. Battery 电池
这个驱动通过/proc/acpi/battery 提供电池信息。如果你有使用电池的移动系统，选 Y。
6.2.3. Button 按钮
这个驱动通过电源、休眠、锁定按钮来提交事件。后台程序读取/proc/acpi/event 来运行用户要求的事件，比如关机。这对软件控制关机是必要的。
6.2.4. Video (ACPI_VIDEO) 视频
提供 ACPI 对主板上的集成显示适配器的扩展支持驱动。详见 ACPI2.0 驱动范例，附录 B，它提供了基本支持，比如定义视频的启动设备、返回 EDID 信息或者设置视频传输等等。

注意这仅仅是文字上的信息而已。它可能（或许不可能）在你的集成显卡设备上运行。
6.2.5. Generic Hotkey (EXPERIMENTAL) 通用热键。
实验中的整合式热键驱动。
如果不清楚，选 N。
6.2.6. 05.03.05、<*> Fan 风扇
这个驱动对 ACPI 风扇设备提供支持，允许用户模式的程序进行风扇的基本控制（开、关、状态显示）
6.2.7. Dock
提供 ACPI Docking station 支持
Docking station 是笔记本的扩展坞，就是用来扩展笔记本电脑功能的底座，通过接口和插槽，它可以连接多种外部设备（驱动器、大屏幕显示器、键盘、打印机、扫 描仪……）。可以弥补轻薄笔记本电脑本身携带附件较少的缺陷，这种设计让用户在办公室里能够享受到台式机一样的便利和舒适，在移动办公时又能发挥笔记本的 便携性。
6.2.8. Processor 处理器
这个驱动以空闲管理者方式给 LINUX 安装 ACPI，使用 ACPI C2 和 C3 处理器状态来节约电能，如果你的系统支持的话。一些 CPU 频率调节的驱动需要这个功能。
6.2.8.1. Thermal Zone
温控区域
ACPI 温控区域驱动。大多数笔记本和台式机支持 ACPI 温控区域。强烈要求你选 Y，否则你的处理器可能会坏掉。
6.2.9. ASUS/Medion Laptop Extras
华硕笔记本扩展支持
6.2.10. IBM ThinkPad Laptop Extras
IBM 笔记本扩展支持
6.2.11. Toshiba Laptop Extras
Toshiba 笔记本扩展支持
6.2.12. (0) Disable ACPI for systems before Jan 1st this year 千年虫
6.2.13. Debug Statements
调试语句
ACPI 驱动可以自定义报告详细的错误信息。选 Y 开启这项功能，这将让你的内核增加 50KB。
6.2.14. ACPI0004,PNP0A05 and PNP0A06 Container Driver
ACPI0004,PNP0A05 和 PNP0A06 容器驱动
这里允许物理上对 CPU 和内存的插入和移除。这对一些系统，比如 NUMA，非常有用，这些系统支持 ACPI 基本的物理拔插。
如果选择 M，这个驱动可以通过命令："modprobe acpi_container"加入。
6.2.15. Smart Battery System
袖珍电池系统
这个驱动对袖珍电池系统提供支持，依赖于 I2C (在选项 Device Drivers ---> I2C support) 。袖珍电池非常古老，也非常稀少，对于今天的 ACPI 支持的电池规范来说。
6.3. APM (Advanced Power Management) BIOS Support
高级电源管理 BIOS 支持。（APM）
ACPI 和 APM 就好比 XP 和 LINUX。我用了 ACPI，这个就只编成模块放着，万一要用到再加模块。不清楚的可以先在机器上用 ps －A｜less 看看有没有这个相关的进程。我的只有 ACPID。

没有认真研究过下面的选项，也不列出来糊弄人了。要是用到 APM 的可以自己研究。
6.4. CPU Frequency scaling
6.4.1. CPU Frequency scaling
CPU 变频控制
CPU 变频控制允许你在运行中改变 CPU 的时钟速度。这是对于节约电能来说是一个不错的主意，因为 CPU 频率越低，它消耗的电能越少。
注意这个驱动不会自动改变 CPU 的时钟速度，你要么允许动态的频率调节器（看下面），要么使用用户工具。
如果不清楚，选 N。
6.4.2. Enable CPUfreq debugging
是否允许调试 CPU 改变主频的功能，如果要调试，还需要在启动时加上参数。cpufreq.debug=<value> 1：变频技术的内核调试 2：变频技术的驱动调试 3：变频技术的调节器调试
6.4.3. CPU frequency translation statistics CPU 频率统计功能
6.4.4. CPU frequency translation statistics details CPU 频率统计功能（详细）
6.4.5. Default CPUFreq governor (performance) 默认的主频调节，圆括号内的是你选择的结果，这里表示以性能为主。
6.4.5.1. performance 性能优先
6.4.5.2. userspace 用户定义，可以设定频率。
6.4.6. 'performance' governor 性能调节器
6.4.7. 'powersave' governor 节约电能调节器。
6.4.8. 'userspace' governor for userspace frequency scaling 用户自定义调节器。
6.4.9. 'ondemand' cpufreq policy governor 自动调节主频。
6.4.10. 'conservative' cpufreq governor 传统方式调节
6.4.11. CPUFreq processor drivers 变频驱动模块
6.4.12. ACPI Processor P-States driver 报告处理器的状态。
6.4.13. AMD Mobile K6-2/K6-3 PowerNow! AMD 移动版 K6 处理器的变频驱动。
6.4.14. AMD Mobile Athlon/Duron PowerNow! AMD 移动版毒龙、雷乌的变频驱动。
6.4.15. Cyrix MediaGX/NatSemi Geode Suspend Modulation Cyrix 处理器的变频驱动。
6.4.16. Intel Enhanced SpeedStep Intel 的移动变频技术支持。
6.4.16.1. Use ACPI tables to decode valid frequency/voltage pairs 使用 BIOS 中的主频／电压参数。
6.4.16.2. Built-in tables for Banias CPUs 迅驰一代的主频／电压参数。
笔记本 ：什么是迅驰技术
2003 年 3 月英特尔正式发布了迅驰移动计算技术，英特尔的迅驰移动计算技术并非以往的处理器、芯片组等 单一产品形式，其代表了一整套移动计算解决方案，迅驰的构成分为三个部分：奔腾 M 处理器、855/915 系列芯片组和英特尔 PRO 无线网上，三项缺一不可 共同组成了迅驰移动计算技术。
奔腾 M 首次改版叫 Dothan
在两年多时间里，迅驰技术经历了一次改版和一次换代。初期迅驰中奔腾 M 处理器的核心代号为 Bannis，采用 130 纳米工艺，1MB 高速二级缓存， 400MHz 前端总线。迅驰首次改版是在 2004 年 5 月，采用 90 纳米工艺 Dothan 核心的奔腾 M 处理器出现，其二级缓存容量提供到 2MB，前端总线仍 为 400MHz，它也就是我们常说的 Dothan 迅驰。首次改版后，Dothan 核心的奔腾 M 处理器迅速占领市场，Bannis 核心产品逐渐退出主流。虽 然市场中

流行着将 Dothan 核心称之为迅驰二代，但英特尔官方并没有给出明确的定义，仍然叫做迅驰。也就是在 Dothan 奔腾 M 推出的同时，英特尔更改 了以主频定义处理器编号的惯例，取而代之的是一系列数字，例如：奔腾 M 715/725 等，它们分别对应 1.5GHz 和 1.6GHz 主频。首次改版中，原 802.11b 无线网卡也改为了支持 802.11b/g 规范，网络传输从 11Mbps 提供至 14Mbps.

新一代迅驰 Sonoma

迅驰的换代是 2005 年 1 月 19 日，英特尔正式发布基于 Sonoma 平台的新一代迅驰移动计算技术，其构成组件中，奔腾 M 处理器升级为 Dothan 核心、 90 纳米工艺、533MHz 前端总线和 2MB 高速二级缓存，处理器编号由奔腾 M 730—770，主频由 1.60GHz 起，最高 2.13GHz。915GM/PM 芯片组让迅驰进入了 PCI-E 时代，其中 915GM 整合了英特尔 GMA900 图形引擎，让非独立显卡笔记本在多媒体性能上有了较大提高。915PM/GM 还支持单通道 DDR333 或双通道 DDR2 400/533MHz 内存，性能提供同时也降低了部分功耗。目前 Sonoma 平台的新一代迅驰渐渐成为市场主流。

6.4.17. Intel Speedstep on ICH-M chipsets (ioport interface) Intel ICH-M 移动南桥芯片组的支持

6.4.18. Intel Pentium 4 clock modulation P4 处理器的时钟模块支持。

6.4.19. Transmeta LongRun Transmeta 处理器的支持。

6.4.20. VIA Cyrix III Longhaul VIA Cyrix 处理器的支持。

6.4.21. shared options

6.4.22. /proc/acpi/processor/../performance interface (deprecated) 从 /proc/acpi/processor/../performance 获得 CPU 的变频信息。

6.4.23. Relaxed speedstep capability checks

不全面检测 Intel Speedstep，有的系统虽然支持 Speedstep 技术，却无法通过全面的检测。

7. Bus options (PCI, PCMCIA, EISA, MCA, ISA) 总线选项

7.1. PCI support

PCI 总线支持（一定要进内核，不能编成模块）

找找你的主板资料，看看你用的是不是 PCI 主板。PCI 是总线系统的名称，是 CPU 用来与其他设备进行通信的通道。其他总线系统有 ISA、EISA、MCA 和 VESA。如果你有 PCI，选 Y。否则，选 N。

7.1.1. PCI access mode (Any)

PCI 访问模式

在 PCI 系统中，BIOS 可以检测 PCI 设备和确定它们的设置。但是，一些老的 PCI 主板有 BIOS 问题，如果这里选上会让系统当机。同时，一些嵌入式的基于 PCI 系统没有任何 BIOS。LINUX 可以在不使用 BIOS 的情况下尝试直接检测 PCI 硬件。

选上这个以后，你可以设定 LINUX 如果检测 PCI 设备。如果你选择"BIOS"，BIOS 会用到。你选 "Direct"， BIOS 不会用到。如果你选"MMConfig"，PCI 加速的 MMCONFIG 会用到。如果你选"Any" ，内核先用 MMCONFIG ，然后 "Direct"，最后才是"BIOS"如果前面的都无法工作。如果不清楚，选"Any"。

7.1.1.1. BIOS

7.1.1.2. MMConfig

7.1.1.3. Direct

7.1.1.4. Any

7.1.2. PCI Express support

PCI Express 支持

这里自动支持 PCI Express 端口总线。用户可以选择 Native Hot-Plug support， Advanced Error Reporting support，
Power Management Event support，Virtual Channel support 4 个选项来支持 PCI Express 端口（启动或者切换）。
我的板是 PCI Express。大家可以用 lshw|less 来看看自己的 PCI 是什么类型。
7.1.2.1. Root Port Advanced Error Reporting support
高级启动错误报告支持。
7.1.3. Message Signaled Interrupts (MSI and MSI-X)
信息信号中断
这允许设备驱动开启 MSI。MSI 允许一个设备用非装订内存写入方式在自己的 PCI 总线中产生一个中断，而不是常规的 IRQ 针脚中断。
在内核启动时，用 'pci=nomsi'选项可以禁用 PCI MSI 中断。这将在整个系统禁用 MSI。
如果不知道怎么做，选 N。
7.1.4. Interrupts on hypertransport devices
高速传输设备中断
允许高速传输设备使用中断。
如果不清楚，选 Y。
7.2. ISA support ISA 总线
看看你主板上是否有 ISA 插槽。ISA 是比较老的总线，现已基本被 PCI 取代。如果你没有老式的 ISA 设备，可以不选这项。不过我估计你的主板上应该会有 ISA 总路。因为我的 INTEL 945 板都还有一路，接老式打印机用的。选上备用。如果你认为你永远都不会用到那一路的话，可以不选。
7.2.1. EISA support
扩展工业标准架构总线（EISA）是为 IBM 微通道开发的项目。EISA 总线可以为 IBM 微通道提供一些特性，如果你在使用很老的卡。目前 EISA 很少用到，已经被 PCI 取代。选 Y，如果你为 ESIA 系统编译内核。否则，选 N。
7.2.2. MCA support
IBM PS/2 上的总线，现已淘汰，建议关闭。微通道总线 IBM 的台式机和笔记本上可能会有这种总线，包括它的 p 系列、e 系列、z 系列机器上都用到了这种总线。
7.2.3. NatSemi SCx200 support
松下的一种半导体处理器的驱动。
7.3. PCCARD (PCMCIA/CardBus) support
一般只有笔记本电脑上才会有 PCMCIA 插槽，如果你是台式机的话，可以不选这一项，然后跳过这一部份。我的 IBM 机器是办公用的，经常会临时接一些乱七八糟的设备。我自己都不知道哪些设备需要什么模块。所以这里我都搞成模块，免得以后接上用不了，又得切换到 XP 下。
7.3.1. PCCard (PCMCIA/CardBus) support
PCI Hotplug Support
PCI 热插拔支持
选 Y，如果你的主板有 PCI 热拔插控制器，这允许你热拔插 PCI 卡。
选 M，将编译为模块，叫做 pci_hotplug。
如果不清楚，选 N。
一般来讲只有服务器上会有热插拔的设备，如果你使用的是台式机，你可以不选择此项并跳过这一部份。

8. Executable file formats

可执行文件格式。

8.1. Kernel support for ELF binaries

ELF 支持

ELF（可执行和可链接格式）是一种用来连接不同架构和操作系统的可执行文件、库函数格式。选 Y，你的内核可以运行 ELF 二进制文件，这也使你的内核增大 13KB。

ELF 现在基本代替了传统的 a.out 格式（QMAGIC and ZMAGIC 用到），因为它是可移植的（可移植不代表它可以直接运行在不同构架和操作系统上），而且建立相关运行库文件非常容易。很多新的可执行文件都用 ELF 格式发布，你在这里当然要选 Y。

8.2. Kernel support for a.out and ECOFF binaries

对 a.out 和 ECOFF 二进制文件的支持

A.out (Assembler.OUTput)是一种二进制文件格式，它用在最早的 UNIX 版本中。LINUX 在 QMAGIC 和 ZMAGIC 两个镜像中使用 A.out，直到它最近被 ELF 取代。ELF 的转变开始于 1995 年。这个选项主要是给研究历史的人提供感兴趣的信息，或者你 要是有那个年代的文件，你需要这个选项。

大多数人在这可以选 N。如果你认为你有可能会用到这个格式，选 M 编译成模块。模块名为 binfmt_aout。如果你系统的关键部件（比如/sbin/init 或者 /lib/ld.so）是 a.out 格式的的，你要在这选 Y。

8.3. Kernel support for MISC binaries

内核对 MISC 二进制文件的支持

如果你在这选 Y，它将可以将 wrapper-driven 二进制格式嵌入内核。当你使用一些程序的解释器时，比如 Java, Python, .NET 或者 Emacs-Lisp，或者当你经常通过 DOS 仿真器运行 DOS 程序时，它将非常有用。当你在这个选项选 Y，你可以简单地通过在 shell 打相应命令运行以上的程序，LINUX 可以自动匹配正确的格 式。

要使用 binfmt_misc 你可能需要挂载它：

mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc

你可以选 M 作为模块，以后再加载，模块名为 binfmt_misc。如果你不知道怎么办，选 Y。

参考网页：触摸屏驱动移植

添加touchscreen驱动到基于linux-2_6_26 的GEC2440 开发板 - yuan_xihua的日志 - 网易博客.mht

参考网页：QT 下载 ftp 地址:

ftp://ftp.trolltech.no/qt/source/

参考网页：UDV 介绍

http://www.mcublog.com/blog/user1/9854/archives/2006/19926.html

第一、什么是 udev?

这篇文章 UDEV Primer 给我们娓娓道来，花点时间预习一下是值得的。当然，不知道 udev 是什么也没关系，把它当个助记符好了，有了下面的上路指南，可以节省很多时间。我们只需要树立一个信念：udev 很简单！嵌入式的 udev 应用尤其简单。

第二、为什么 udev 要取代 devfs?

这是生产关系适应生产力的需要，udev 好，devfs 坏，用好的不用坏的。

udev 是硬件平台无关的，属于 user space 的进程，它脱离驱动层的关联而建立在操作系统之上，基于这种设计实现，我们可以随时修改及删除/dev 下的设备文件名称和指向，随心所欲地按照我们的愿望安排和管理设备文件系统，而完成如此灵活的功能只需要简单地修改 udev 的配置文件即可，无需重新启动操作系统。udev 已经使得我们对设备的管理如探囊取物般轻松自如。

第三、如何得到 udev?

udev的主页在这里：http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html
我们按照下面的步骤来生成udev的工具程序，以arm-linux为例：
1、wget http://www.us.kernel.org/pub/linux/utils/kernel/hotplug/udev-100.tar.bz2
2、tar xjf udev-100.tar.bz2
3、cd udev-100 编辑Makefile，查找CROSS_COMPILE，修改CROSS_COMPILE ?= arm-linux-
4、make

没有什么意外的话当前目录下生成 udev,udevcontrol,udevd,udevinfo,udevmonitor,udevsettle,udevstart,udevtest,udevtrigger 九个工具程序，在嵌入式系统里，我们只需要 udevd 和 udevstart 就能使 udev 工作得很好，其他工具则帮助我们完成 udev 的信息察看、事件捕捉或者更高级的操作。

另外一个方法是直接使用debian提供的已编译好的二进制包，美中不足的是版本老了一些。
1、wget http://ftp.us.debian.org/debian/pool/main/u/udev/udev_0.056-3_arm.deb
2、ar -xf udev_0.056-3_arm.deb
3、tar xzf data.tar.gz

在 sbin 目录里就有我们需要的 udevd 和 udevstart 工具程序。

建议大家采用第一种方式生成的 udevd 和 udevstart。为什么要用最新 udev 呢？新的强，旧的弱，用强的不用弱的。

第四、如何配置 udev?

首先，udev 需要内核 sysfs 和 tmpfs 的支持，sysfs 为 udev 提供设备入口和 uevent 通道，tmpfs 为 udev 设备文件提供存放空间，也就是说，在上电之前系统上是没有足够的设备文件可用的，我们需要一些技巧让 kernel 先引导起来。

由于在 kernel 启动未完成以前我们的设备文件不可用，如果使用 mtd 设备作为 rootfs 的挂载点，这个时候/dev/mtdblock
是不存在的，我们无法让 kernel 找到 rootfs，kernel 只好停在那里惊慌。
这个问题我们可以通过给 kernel 传递设备号的方式来解决，在 linux 系统中，mtdblock 的主设备号是 31，part 号从 0 开始，那么以前的/dev/mtdblock/3 就等同于 31:03，以次类推，所以我们只需要修改 bootloader 传给 kernel 的 cmd line 参数，使 root=31:03，就可以让 kernel 在 udevd 未起来之前成功的找到 rootfs。
O.K.下一个问题。

其次，需要做的工作就是重新生成 rootfs，把 udevd 和 udevstart 复制到/sbin 目录。然后我们需要在/etc/下为 udev 建立设备规则，这可以说是 udev 最为复杂的一步。这篇文章提供了最完整的指导：Writing udev rules 文中描述的复杂规则我们可以暂时不用去理会，上路指南将带领我们轻松穿过这片迷雾。这里提供一个由简入繁的方法，对于嵌入式系统，这样做可以一劳永逸。

1、在前面用到的 udev-100 目录里，有一个 etc 目录，里面放着的 udev 目录包含了 udev 设备规则的详细样例文本。为了简单而又简洁，我们只需要用到 etc/udev/udev.conf 这个文件，在我们的 rootfs/etc 下建立一个 udev 目录，把它复制过去，这个文件很简单，除了注释只有一行，是用来配置日志信息的，嵌入式系统也许用不上日志，但是 udevd 需要检查这个文件。

2、在 rootfs/etc/udev 下建立一个 rules.d 目录，生成一个空的配置文件 touch etc/udev/rules.d/udev.conf。然后我们来编辑这个文件并向它写入以下配置项：

```
##################################################
# vc devices
KERNEL=="tty[0-9]*", NAME="vc/%n"

# block devices
KERNEL=="loop[0-9]*", NAME="loop/%n"

# mtd devices
KERNEL=="mtd[0-9]*", NAME="mtd/%n"
KERNEL=="mtdblock*", NAME="mtdblock/%n"

# input devices
KERNEL=="mice" NAME="input/%k"
KERNEL=="mouse[0-9]*", NAME="input/%k"
KERNEL=="ts[0-9]*", NAME="input/%k"
KERNEL=="event[0-9]*", NAME="input/%k"

# misc devices
KERNEL=="apm_bios", NAME="misc/%k"
KERNEL=="rtc", NAME="misc/%k"
##################################################
```

保存它，我们的设备文件系统基本上就可以了，udevd 和 udevstart 会自动分析这个文件。

3、为了使 udevd 在 kernel 起来后能够自动运行，我们在 rootfs/etc/init.d/rcS 中增加以下几行：

```
###################################
/bin/mount -t tmpfs tmpfs /dev

echo "Starting udevd..."
/sbin/udevd --daemon
/sbin/udevstart
###################################
```

4、重新生成 rootfs，烧写到 flash 指定的 rootfs part 中。

5、如果需要动态改变设备规则，可以把 etc/udev 放到 jffs 或 yaffs part，以备修改，根据需求而定，可以随时扩充 udev.conf 中的配置项。

－－ 以下是我的问题

Starting udevd...

/etc/init.d/rcS: /etc/init.d/rcS: 7: /sbin/udevd: not found

/etc/init.d/rcS: /etc/init.d/rcS: 8: /sbin/udevstart: not found

可是在/sbin/下确实存在这两个文件，而且/etc/udev/下面的文件我也已经建好。

我在进入 shell 后，手动运行：

/sbin $ udevd --daemon

/bin/sh: udevd: not found

还是一样的情况，请高人指点迷津

相应的库文件都没有当然会提示错误了，还有一个重要问题他没提到，那就是在/etc/init.d/rcS 下要有这句 mount -t sysfs sysfs /sys 否则无法启动系统。

原文出处 http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ
中文翻译 王旭 http://gnawux.blogchina.com
本文档遵循 GPL 2 及以后版本发布，修改、发布请保持许可证不变

问: udev 是什么? 它的目的何在?
答: 看看那篇 OLS 2003 上的有关 udev 的文章吧，可以在 docs 目录里找到，也能在这里找到:

OLS 2003 上还有一个关于 udev 的幻灯片，可以在这里找到:

问: udev 和 devfs 是什么关系
答: udev 完全在用户态 (userspace) 工作，利用设备加入或移除时内核所发送的 hotplug 事件 (event) 来工作。关于设备的详细信息是由内核输出 (export) 到位于 /sys 的 sysfs 文件系统的。所有的设备命名策略、权限控制和事件处理都是在用户态下完成的。与此相反，devfs 是作为内核的一部分工作的。

问: 如果 udev 不能完成所有 devfs 的工作的话，为什么把 devfs 标记为 OBSOLETE/removed?
答: 引用 Al Viro (Linux VFS 内核维护者):
- devfs 所做的工作被确信可以在用户态来完成。
- devfs 被加入内核之时，大家寄望它的质量可以迎头赶上。
- devfs 被发现了一些可修复和无法修复的 bug。
- 对于可修复的 bug，几个月前就已经被修复了，其维护者认为一切良好。

- 对于后者，同样是相当常一段时间以来没有改观了。
- devfs 的维护者和作者对它感到失望并且已经停止了对代码的维护工作。

问: 但是当一个并不存在的 /dev 节点被打开的时候，udev 并不能如 devfs 一样自动加载驱动程序。
答: 的确如此，但 Linux 的设计是在设备被发现的时候加载模块，而不是当它被访问的时候。

问: 不过等等，我确实希望 udev 可以在不存在的节点被打开的时候自动加载驱动。这是我使用 devfs 的唯一原因了。给 udev 增加这个功能吧。
答: 不，udev 是用来管理 /dev 的，不是用来加载内核驱动的。

问: 嗨，求你们了。这不难做到的。
答: 这么个功能对于一个配置正确的计算机是多余的。系统中所有的设备都应该产生 hotplug 事件、加载恰当的驱动，而 udev 将会注意到这点并且为它创建对应的设备节点。如果你不想让所有的设备驱动停留在内存之中，应该使用其它东西来管理你的模块 (如脚本, modules.conf, 等等) 这不是 udev 的工作。

问: 但是我真的喜欢那个功能，还是加上吧
答: devfs 用的方法导致了大量无用的 modprobe 尝试，以此程序探测设备是否存在。每个试探性探测都新建一个运行 modprobe 的进程，而几乎所有这些都是无用的。

问: 我喜欢 devfs 的设备文件命名方式，udev 可以这样命名么?
答: 可以，udev 可以使用 /dev 的命名策略来创建节点。通过一个配置文件，可以把内核缺省的名字映射到 devfs 的名字。可以看看 udev 中带的 udev.rules.devfs 文件。
注意: devfs 的命名方式是不被建议并且不被官方支持的，因为它所用的简单枚举设备的方式在设备可能被随时加入或删除的情况下确实是一个比较笨的方法。这些编号代给你的将只有麻烦，而并不能用来确定设备。看看那个永久性磁盘 (persistent disk) 的规则就知道如何在用户态下正确的做这件事，而不是傻傻地列出设备。

问: udev 可以为哪些设备创建节点?
答: 所有在 sysfs 中显示的设备都可以由 udev 来创建节点。如果内核中增加了其它设备的支持，udev 也就自动地可以为它们工作了。现在所有的块设备都在被支持之列，大部分的主字符设备也是被支持的。内核开发者们正致力于让所有的字符设备都被支持。可以到 linux-kernel 邮件列表上寻找补丁或是查看补丁的状态。

问: udev 是否会去掉匿名设备数量的限制?
答: udev 完全工作于用户态。如果内核支持了更多的匿名设备，udev 就会支持。

问: udev 是否会支持符号链接?
答: udev 现在就支持符号链接，每个设备节点拥有多个符号链接也是被支持的。

问: udev 如何处理 /dev 文件系统?

答: 建议使用一个每次启动系统的时候重新创建的 tmpfs 作为 /dev 的文件系统。不过实际上 udev 并不关心那种文件系统在被使用。

问: 在 init 运行之前，udev 如何处理设备?
答: udev 可以被放入 initramfs 之中，并在每个设备被发现的时候运行。也可以让 udev 工作在一个真的根分区被加载之后根据 /sys 的内容创建的初始 /dev 目录之中。

问: 我是否可以利用 udev 在一个 USB 设备被加载的时候自动加载上这个设备?
答: 技术上讲是可以的，但是 udev 不是用于这个工作的。所有的主流发布版 (distro) 都包含了 HAL (http://freedesktop.org/wiki/Software_2fhal) 用于这个工作，它也是专门用于监视设备变更的，并且集成进入了桌面软件。

换个角度说，这可以简单的通过 fstab 来实现:
/dev/disk/by-label/PENDRIVE /media/PENDRIVE vfat user,noauto 0 0

这样，用户可以用如下命令来访问设备:
$mount /media/PENDRIVE
同样不需要管理员权限，但却拥有了设备的全部访问权限。使用永久性磁盘链接 (label, uuid) 将可以指定同一设备，无论其实际上的内核名字是什么。

问: 有什么我需要注意的安全问题么?
答: 当使用动态设备编号的时候，一个给定的主/从设备号可能在不同时间对应不同的设备，如果一个用户拥有对这个节点的访问权限，并且可以创建一个到这个节点的硬链接，他就可以如此得到一个这个设备节点的拷贝。当设备被移除之后，udev 删除了设备节点，但硬链接依然存在。如果这个设备节点之后被重新使用不同的访问权限被创建的时候，其硬链接仍然可以使用先前的访问权限来访问。
(同样的问题也存在在使用 PAM 改变访问权限的 login 上。)

简单的解决方案就是通过把 /dev 放在 tmpfs 这样的单独的文件系统之上来防止建立硬链接。

问: 我有其他的关于 udev 的问题，我应该问谁?
答: linux-hotplug-devel 正是问这些的地方。邮件列表的地址是
linux-hotplug-devel@lists.sourceforge.net
加入邮件列表的相关信息可以在如下地址找到

邮件列表的上的既往讨论记录可以在下面地址找到

参考网页：如何编写 Linux 设备驱动程序

http://www.mcublog.com/blog/user1/9854/archives/2006/20206.html

Linux是Unix操作系统的一种变种，在Linux下编写驱动程序的原理和思想完全类似于其他的Unix系统，但它dos或window环境下的驱动程序有很大的区别。在Linux环境下设计驱动程序，思想简洁，操作方便，功能也很强大，但是支持函数少，只能依赖kernel中的函数，有些常用的操作要自己来编写，而且调试也不方便。本人这几周来为实验室自行研制的一块多媒体卡编制了驱动程序，获得了一些经验，愿与Linux fans共享，有不当之处，请予指正。

以下的一些文字主要来源于 khg，johnsonm 的 Write linux device driver，Brennan's Guide to Inline Assembly，The Linux A-Z，还有清华 BBS 上的有关 device driver 的一些资料. 这些资料有的已经过时，有的还有一些错误，我依据自己的试验结果进行了修正.

一、Linux device driver 的概念

系统调用是操作系统内核和应用程序之间的接口，设备驱动程序是操作系统内核和机器硬件之间的接口.设备驱动程序为应用程序屏蔽了硬件的细节，这样在应用程序看来，硬件设备只是一个设备文件， 应用程序可以象操作普通文件一样对硬件设备进行操作.设备驱动程序是内核的一部分，它完成以下的功能:

1.对设备初始化和释放.

2.把数据从内核传送到硬件和从硬件读取数据.

3.读取应用程序传送给设备文件的数据和回送应用程序请求的数据.

4.检测和处理设备出现的错误.

在 Linux 操作系统下有两类主要的设备文件类型，一种是字符设备，另一种是块设备.字符设备和块设备的主要区别是:在对字符设备发出读/写请求时，实际的硬件 I/O 一般就紧接着发生了，块设备则不然，它利用一块系统内存作缓冲区，当用户进程对设备请求能满足用户的要求，就返回请求的数据，如果不能，就调用请求函数来进行实际的 I/O 操作.块设备是主要针对磁盘等慢速设备设计的，以免耗费过多的 CPU 时间来等待.

已经提到，用户进程是通过设备文件来与实际的硬件打交道.每个设备文件都都有其文件属性(c/b)，表示是字符设备还蠓强橹璞?另外每个文件都有两个设备号，第一个是主设备号，标识驱动程序，第二个是从设备号，标识使用同一个设备驱动程序的不同的硬件设备，比如有两个软盘，就可以用从设备号来区分他们.设备文件的的主设备号必须与设备驱动程序在登记时申请的主设备号一致，否则用户进程将无法访问到驱动程序.

最后必须提到的是，在用户进程调用驱动程序时，系统进入核心态，这时不再是抢先式调度.也就是说，系统必须在你的驱动程序的子函数返回后才能进行其他的工作.如果你的驱动程序陷入死循环，不幸的是你只有重新启动机器了，然后就是漫长的 fsck.//hehe

读/写时，它首先察看缓冲区的内容，如果缓冲区的数据

如何编写 Linux 操作系统下的设备驱动程序 -6-28 18:15:38

二、实例剖析

我们来写一个最简单的字符设备驱动程序。虽然它什么也不做，但是通过它可以了解 Linux 的设备驱动程序的工作原理.把下面的 C 代码输入机器，你就会获得一个真正的设备驱动程序.不过我的 kernel 是 2.0.34，在低版本的 kernel 上可能会出现问题，我还没测试过.//xixi

```
#define __NO_VERSION__
#i nclude <linux/modules.h>
#i nclude <linux/version.h>

char kernel_version [] = UTS_RELEASE;
```

这一段定义了一些版本信息，虽然用处不是很大，但也必不可少.Johnsonm 说所有的驱动程序的开头都要包含<linux/config.h>，但我看倒是未必.

由于用户进程是通过设备文件同硬件打交道，对设备文件的操作方式不外乎就是一些系统调用，如 open，read，write，close....， 注意，不是 fopen， fread，但是如何把系统调用和驱动程序关联起来呢?这需要了解一个非常关键的数据结构:

```
struct file_operations {

int (*seek) (struct inode * , struct file *,  off_t , int);
int (*read) (struct inode * , struct file *,  char , int);
int (*write) (struct inode * , struct file *,  off_t , int);
int (*readdir) (struct inode * , struct file *,  struct dirent * , int);
int (*select) (struct inode * , struct file *,  int , select_table *);
int (*ioctl) (struct inode * , struct file *,  unsined int , unsigned long);
int (*mmap) (struct inode * , struct file *,  struct vm_area_struct *);
int (*open) (struct inode * , struct file *);
int (*release) (struct inode * , struct file *);
int (*fsync) (struct inode * , struct file *);
int (*fasync) (struct inode * , struct file *, int);
int (*check_media_change) (struct inode * , struct file *);
int (*revalidate) (dev_t dev);
}
```

这个结构的每一个成员的名字都对应着一个系统调用.用户进程利用系统调用在对设备文件进行诸如 read/write 操作时，系统调用通过设备文件的主设备号找到相应的设备驱动程序，然后读取这个数据结构相应的函数指针，接着把控制权交给该函数.这是 linux 的设备驱动程序工作的基本原理.既然是这样，则编写设备驱动程序的主要工作就是编写子函数，并填充 file_operations 的各个域.

相当简单，不是吗?

下面就开始写子程序.

```
#i nclude <linux/types.h>
#i nclude <linux/fs.h>
#i nclude <linux/mm.h>
#i nclude <linux/errno.h>
#i nclude <asm/segment.h>
unsigned int test_major = 0;

static int read_test(struct inode *node, struct file *file,
char *buf, int count)
{
```

```
int left;

if (verify_area(VERIFY_WRITE, buf, count) == -EFAULT )
return -EFAULT;

for(left = count ; left > 0 ; left--)
{
__put_user(1, buf, 1);
buf++;
}
return count;
}
```

这个函数是为 read 调用准备的.当调用 read 时，read_test()被调用，它把用户的缓冲区全部写 1.buf 是 read 调用的一个参数.它是用户进程空间的一个地址.但是在 read_test 被调用时，系统进入核心态.所以不能使用 buf 这个地址，必须用__put_user()，这是 kernel 提供的一个函数，用于向用户传送数据.另外还有很多类似功能的函数.请参考.在向用户空间拷贝数据之前，必须验证 buf 是否可用。

这就用到函数 verify_area.

```
static int write_tibet(struct inode *inode, struct file *file,
const char *buf, int count)
{
return count;
}

static int open_tibet(struct inode *inode, struct file *file )
{
MOD_INC_USE_COUNT;
return 0;
}

static void release_tibet(struct inode *inode, struct file *file )
{
MOD_DEC_USE_COUNT;
}
```

这几个函数都是空操作.实际调用发生时什么也不做，他们仅仅为下面的结构提供函数指针。

```
struct file_operations test_fops = {
NULL,
read_test,
```

```
write_test,
NULL,  /* test_readdir */
NULL,
NULL,  /* test_ioctl */
NULL,  /* test_mmap */
open_test,
release_test,  NULL,  /* test_fsync */
NULL,  /* test_fasync */
/* nothing more,  fill with NULLs */
};
```

设备驱动程序的主体可以说是写好了。现在要把驱动程序嵌入内核。驱动程序可以按照两种方式编译。一种是编译进 kernel，另一种是编译成模块(modules)，如果编译进内核的话，会增加内核的大小，还要改动内核的源文件，而且不能动态的卸载，不利于调试，所以推荐使用模块方式。

```
int init_module(void) WWW.GONGWU.COM.CN 2005-6-28 18:15:38
{
int result;

result = register_chrdev(0,  "test",  &test_fops);

if (result < 0) {
printk(KERN_INFO "test: can't get major number\n");
return result;
}

if (test_major == 0) test_major = result; /* dynamic */
return 0;
}
```

在用 insmod 命令将编译好的模块调入内存时，init_module 函数被调用。在这里，init_module 只做了一件事，就是向系统的字符设备表登记了一个字符设备。register_chrdev 需要三个参数，参数一是希望获得的设备号，如果是零的话，系统将选择一个没有被占用的设备号返回。参数二是设备文件名，参数三用来登记驱动程序实际执行操作的函数的指针。

如果登记成功，返回设备的主设备号，不成功，返回一个负值。

```
void cleanup_module(void)
{
unregister_chrdev(test_major,  "test");
}
```

在用 rmmod 卸载模块时，cleanup_module 函数被调用，它释放字符设备 test 在系统字符设备表中占有的表项。

一个极其简单的字符设备可以说写好了，文件名就叫 test.c 吧。

下面编译

$ gcc -O2 -DMODULE -D__KERNEL__ -c test.c

得到文件 test.o 就是一个设备驱动程序。

如果设备驱动程序有多个文件，把每个文件按上面的命令行编译，然后

ld -r file1.o file2.o -o modulename.

驱动程序已经编译好了，现在把它安装到系统中去。

$ insmod -f test.o

如果安装成功，在/proc/devices 文件中就可以看到设备 test，并可以看到它的主设备号。

要卸载的话，运行

$ rmmod test

下一步要创建设备文件。

mknod /dev/test c major minor

c 是指字符设备，major 是主设备号，就是在/proc/devices 里看到的。

用 shell 命令

$ cat /proc/devices | awk "}"

就可以获得主设备号，可以把上面的命令行加入你的 shell script 中去。

minor 是从设备号，设置成 0 就可以了。

我们现在可以通过设备文件来访问我们的驱动程序。写一个小小的测试程序。

```
#i nclude <stdio.h>
#i nclude <sys/types.h>
#i nclude <sys/stat.h>
#i nclude <fcntl.h>

main()
{
```

```
int testdev;
int i;
char buf[10];

testdev = open("/dev/test", O_RDWR);

if ( testdev == -1 )
{
printf("Cann't open file \n");
exit(0);
}

read(testdev, buf, 10);

for (i = 0; i < 10;i++)
printf("%d\n", buf[i]);

close(testdev);
}
```

编译运行，看看是不是打印出全 1 ？

以上只是一个简单的演示。真正实用的驱动程序要复杂的多，要处理如中断，DMA，I/O port 等问题。这些才是真正的难点。请看下节，实际情况的处理。

如何编写 Linux 操作系统下的设备驱动程序

三、设备驱动程序中的一些具体问题

1. I/O Port.

和硬件打交道离不开 I/O Port，老的 ISA 设备经常是占用实际的 I/O 端口，在 linux 下，操作系统没有对 I/O 口屏蔽，也就是说，任何驱动程序都可对任意的 I/O 口操作，这样就很容易引起混乱。每个驱动程序应该自己避免误用端口。

有两个重要的 kernel 函数可以保证驱动程序做到这一点。

1）check_region(int io_port， int off_set)

这个函数察看系统的 I/O 表，看是否有别的驱动程序占用某一段 I/O 口。

参数 1：io 端口的基地址，

参数 2：io 端口占用的范围。

返回值：0 没有占用， 非 0，已经被占用。

2）request_region(int io_port， int off_set，char *devname)

如果这段 I/O 端口没有被占用，在我们的驱动程序中就可以使用它。在使用之前，必须向系统登记，以防止被其他程序占用。登记后，在/proc/ioports 文件中可以看到你登记的 io 口。

参数 1：io 端口的基地址。

参数 2：io 端口占用的范围。

参数 3：使用这段 io 地址的设备名。

在对 I/O 口登记后，就可以放心地用 inb()， outb()之类的函来访问了。

在一些 pci 设备中，I/O 端口被映射到一段内存中去，要访问这些端口就相当于访问一段内存。经常性的，我们要获得一块内存的物理地址。在 dos 环境下，（之所以不说是 dos 操作系统是因为我认为 DOS 根本就不是一个操作系统，它实在是太简单，太不安全了）只要用段：偏移就可以了。在 window95 中，95ddk 提供了一个 vmm 调用 _MapLinearToPhys，用以把线性地址转化为物理地址。但在 Linux 中是怎样做的呢？

2.内存操作

在设备驱动程序中动态开辟内存，不是用 malloc，而是 kmalloc，或者用 get_free_pages 直接申请页。释放内存用的是 kfree，或 free_pages. 请注意，kmalloc 等函数返回的是物理地址！而 malloc 等返回的是线性地址！关于 kmalloc 返回的是物理地址这一点本人有点不太明白：既然从线性地址到物理地址的转换是由 386cpu 硬件完成的，那样汇编指令的操作数应该是线性地址，驱动程序同样也不能直接使用物理地址而是线性地址。但是事实上 kmalloc 返回的确实是物理地址，而且也可以直接通过它访问实际的 RAM，我想这样可以由两种解释，一种是在核心态禁止分页，但是这好像不太现实；另一种是 linux 的页目录和页表项设计得正好使得物理地址等同于线性地址。我的想法不知对不对，还请高手指教。

言归正传，要注意 kmalloc 最大只能开辟 128k-16，16 个字节是被页描述符结构占用了。kmalloc 用法参见 khg.

内存映射的 I/O 口，寄存器或者是硬件设备的 RAM(如显存)一般占用 F0000000 以上的地址空间。在驱动程序中不能直接访问，要通过 kernel 函数 vremap 获得重新映射以后的地址。

另外，很多硬件需要一块比较大的连续内存用作 DMA 传送。这块内存需要一直驻留在内存，不能被交换到文件中去。但是 kmalloc 最多只能开辟 128k 的内存。

这可以通过牺牲一些系统内存的方法来解决。

具体做法是：比如说你的机器由 32M 的内存，在 lilo.conf 的启动参数中加上 mem=30M，这样 linux 就认为你的机器只有 30M 的内存，剩下的 2M 内存在 vremap 之后就可以为 DMA 所用了。

请记住，用 vremap 映射后的内存，不用时应用 unremap 释放，否则会浪费页表。

3.中断处理

同处理 I/O 端口一样，要使用一个中断，必须先向系统登记。

```
int request_irq(unsigned int irq ,
void(*handle)(int, void *, struct pt_regs *),
unsigned int long flags,
const char *device);
```

irq: 是要申请的中断。

handle：中断处理函数指针。

flags：SA_INTERRUPT 请求一个快速中断，0 正常中断。

device：设备名。

如果登记成功，返回 0，这时在/proc/interrupts 文件中可以看你请求的中断。

4.一些常见的问题。

对硬件操作，有时时序很重要。但是如果用 C 语言写一些低级的硬件操作的话，gcc 往往会对你的程序进行优化，这样时序就错掉了。如果用汇编写呢，gcc 同样会对汇编代码进行优化，除非你用 volatile 关键字修饰。最保险的办法是禁止优化。这当然只能对一部分你自己编写的代码。如果对所有的代码都不优化，你会发现驱动程序根本无法装载。这是因为在编译驱动程序时要用到 gcc 的一些扩展特性，而这些扩展特性必须在加了优化选项之后才能体现出来。