    看完了前面说的几本书之后，对Linux Kernel和Android有一定的认识了，是不是心里蠢蠢欲动，想小试牛刀自己编译一把Android源代码了呢？一直习惯使用Windows系统，而Android源代码是不支持在Windows上编译上，于是决定使用虚拟机安装Ubuntu，然后下载、编译和安装Android源代码。

     一. 环境准备。

     1. 磁盘空间预留20G左右，内存3G，因为一边要跑主机，一边要跑虚拟机，内存要求还是比较高的，这样才会比较流畅。

     2. 安装VMWare 7.1.4。我的操作系统是Win7，VMWare的版本要新一点的，旧版本的VMWare在网络支持上比较差，由于要在虚拟机上下载Android源代码，没有网络是万万不行的。

     3. 安装好VMWare后，接下来就安装Ubuntu系统了。我选择目前最新的版本ubuntu-11.04-alternate-i386，从网上查到的资料说，要编译Android源代码，Ubuntu的最低版本是8.04。下载好后，安装时采用一直默认安装即可。

     4. 安装Git工具。Android源代码采用Git工具来管理，与SVN相比，这是一种分布式的源代码管理工具，而SVN是集中式的源代码管理工具。要安装Git工具，在Ubuntu上执行以下命令即可：

**USER-NAME@MACHINE-NAME:~$ sudo apt-get install git-core gnupg**

5. 安装Java SDK。在Ubuntu上执行以下命令：

**USER-NAME@MACHINE-NAME:~$**

sudo add-apt-repository ppa:ferramroberto/java

**USER-NAME@MACHINE-NAME:~$**

sudo apt-get update

**USER-NAME@MACHINE-NAME:~$**

sudo apt-get install sun-java6-jre sun-java6-plugin

**USER-NAME@MACHINE-NAME:~$**

sudo apt-get install sun-java6-jdk

     6. 依赖的其它包。在Ubuntu上执行以下命令：

**USER-NAME@MACHINE-NAME:~$ sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl**

     7. 调试工具。在Ubuntu上执行以下命令：

**USER-NAME@MACHINE-NAME:~$ sudo apt-get install valgrind**

二. 下载Android源代码工程。

     1. 下载repo工具。在Ubuntu上执行以下命令：

     USER-NAME@MACHINE-NAME:~$ wget <http://android.git.kernel.org/repo>

     USER-NAME@MACHINE-NAME:~$ chmod 777 repo

     USER-NAME@MACHINE-NAME:~$ cp repo /bin/

     2. 下载Android最新版本源代码。在Ubuntu上执行以下命令：

**USER-NAME@MACHINE-NAME:~$ mkdir Android**

**USER-NAME@MACHINE-NAME:~$ cd Android**

**USER-NAME@MACHINE-NAME:~/Android$ repo init -u git://**[**android.git.kernel.org/platform/manifest.git**](http://android.git.kernel.org/platform/manifest.git)

**USER-NAME@MACHINE-NAME:~/Android$ repo sync**

经过漫长的等待（我下载了两三天）后，就可以把Android源代码下载下来了。其间可能还有经历下载中断的情况，这时只要重新执行repo sync就可以了。

     三. 编译Android源代码。

     1. 编译。在Android目录下执行以下命令：

**USER-NAME@MACHINE-NAME:~/Android$ make**

第一次编译要等待比较久的时间，编译成功后，可以看到下面的输出：

**Target system fs image:    out/target/product/generic/obj/PACKAGING/systemimage\_intermediates/system.img**

**Install system fs image: out/target/product/generic/system.img**

**Target ram disk: out/target/product/generic/ramdisk.img**

**Target userdata fs image: out/target/product/generic/userdata.img**

**Installed file list: out/target/product/generic/installed-files.txt**

2. 编译过程中可能会遇到的问题。

***问题一：You are attempting to build on a 32-bit system.***

     两个地方需要个修改：

     1）修改build/core目录下的main.mk文件：

*ifeq ($(BUILD\_OS),linux)*

*build\_arch := $(shell uname -m)*

***#Change the following line for building on a 32-bit system.***

***#ifneq (64,$(findstring 64,$(build\_arch)))***

***ifneq (i686,$(findstring i686,$(build\_arch)))***

*$(warning \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)*

*$(warning You are attempting to build on a 32-bit system.)*

*$(warning Only 64-bit build environments are supported beyond froyo/2.2.)*

     2）找到下列文件：

     /external/clearsilver/cgi/Android.mk

     /external/clearsilver/cs/Android.mk

     /external/clearsilver/java-jni/Android.mk

     /external/clearsilver/util/Android.mk

     修改LOCAL\_CFLAGS和LOCAL\_LDFLAGS变量：

*# This forces a 64-bit build for Java6*

*# Change the following two lines for building on a 32-bit system.*

***# LOCAL\_CFLAGS += -m64***

***# LOCAL\_LDFLAGS += -m64***

***LOCAL\_CFLAGS += -m32***

***LOCAL\_LDFLAGS += -m32***

***问题二：Undefined reference to `\_\_dso\_handle'***

     external/stlport/src/monetary.cpp:39: undefined reference to `\_\_dso\_handle'

out/target/product/vm/obj/SHARED\_LIBRARIES/libstlport\_intermediates/src/locale.o: In function `\_\_static\_initialization\_and\_destruction\_0':

     external/stlport/src/locale.cpp:29: undefined reference to `\_\_dso\_handle'

out/target/product/vm/obj/SHARED\_LIBRARIES/libstlport\_intermediates/src/locale\_impl.o: In function `\_\_static\_initialization\_and\_destruction\_0':

     external/stlport/src/locale\_impl.cpp:31: undefined reference to `\_\_dso\_handle'

out/target/product/vm/obj/SHARED\_LIBRARIES/libstlport\_intermediates/src/locale\_impl.o: In function `std::\_Locale\_impl::make\_classic\_locale()':

     external/stlport/src/locale\_impl.cpp:670: undefined reference to `\_\_dso\_handle'

     external/stlport/src/locale\_impl.cpp:667: undefined reference to `\_\_dso\_handle'

out/target/product/vm/obj/SHARED\_LIBRARIES/libstlport\_intermediates/src/locale\_impl.o:external/stlport/src/locale\_impl.cpp:604: more undefined

     references to `\_\_dso\_handle' follow

     collect2: ld returned 1 exit status

     修改external/stlport/dll\_main.cpp，加入以下声明：

***extern "C" {***

***void \* \_\_dso\_handle = 0;***

***}***

四. 编译SDK，这一步是可选的。

     1. 编译。执行以下命令：

**USER-NAME@MACHINE-NAME:~/Android$ make sdk**

2. 编译过程中可能会遇到的问题。

***问题一：找不到bios.bin和vgabios-cirrus.bin文件***

     couldn't locate source file: usr/share/pc-bios/bios.bin

     couldn't locate source file: usr/share/pc-bios/vgabios-cirrus.bin

     注意，这里的usr/share目录指的是~/Android/out/host/linux-x86目录下的usr/share目录，修改办法是复制~/Android/prebuilt/common下的pc-bios文件夹到~/Android/out/host/linux-x86/usr/share即可：

**USER-NAME@MACHINE-NAME:~/Android$ cp ~/Android/prebuilt/common/pc-bios ~/Android/out/host/linux-x86/usr/share**

***问题二：找不到ddmlib-tests.jar、 ninepath-tests.jar 、common-tests.jar 和sdkuilib-tests.jar文件***

    在~/Android/out/host/linux-x86/framework这个目录下，可以找到以下几个文件common.jar、ddmlib.jar、ninepatch.jar、sdkuilib.jar这四个文件，然后将它们分别复制一份，并重命名，命名的原则很简单，就是在原有的名字后面跟上－tests即可。

    五. 安装编译好的Android镜像到模拟器上。

    1. 设置环境变量：

**USER-NAME@MACHINE-NAME:~/Android$ export PATH=$PATH:~/Android/out/host/linux-x86/bin**

**USER-NAME@MACHINE-NAME:~/Android$ export ANDROID\_PRODUCT\_OUT=~/Android/out/target/product/generic**

其中，~/Android/out/host/linux-x86/bin有我们要执行的emulator命令，而~/Android/out/target/product/generic是Android镜像存放目录，下面执行emulator命令时会用到。

    2. 运行模拟器。

**USER-NAME@MACHINE-NAME:~/Android$ emulator**

模拟器运行需要四个文件，分别是Linux Kernel镜像zImage和Android镜像文件system.img、userdata.img和ramdisk.img。执行emulator命令时，如果不带任何参数，则Linux Kernel镜像默认使用~/Android/prebuilt/android-arm/kernel目录下的kernel-qemu文件，而Android镜像文件则默认使用ANDROID\_PRODUCT\_OUT目录下的system.img、userdata.img和ramdisk.img，也就是我们刚刚编译出来的镜像问题。

    当然，我们也可以以指定的镜像文件来运行模拟器，即运行emulator时，即：

**USER-NAME@MACHINE-NAME:~/Android$ emulator -kernel ./prebuilt/android-arm/kernel/kernel-qemu -sysdir ./out/target/product/generic -system system.img -data userdata.img -ramdisk ramdisk.img**

到这里，我们就可以在模拟器上运行我们自己编译的Android镜像文件了，是不是很酷呢？但是注意，这里说的Android镜像文件，只是包括system.img、userdata.img和ramdisk.img这三个文件，而Linux Kernel镜像用的是Android为我们预编译好的kernel-qemu镜像。那么，有没有办法使用我们自己编译的Linux Kernel镜像呢？答案上肯定的，这样我们就可以完全DIY自己的Android系统了！我将在下一篇文章描述如果编译自己的Linux Kernel镜像，敬请期待~



**PS：主线上最新源代码是不稳定版本，使用过程可能会有问题**