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
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