Porting Libnfc to Android

Hendri Appelmelk
Department of Computer Science
Amsterdam, The Netherlands

February 17, 2011

1 Getting the environment to work

This Appendix is a step-by-step guide to prepare a Nexus One phone to operate with an external NFC-reader. In order to port complete this guide, make sure that the next items are available:

- Nexus One mobile phone with Android Froyo 2.2.1 (build FRG83D);
- Ubuntu 10.04 as an Operating System on the host computer;
- External Arygon ADRB NFC-reader;
- USB dual cable type A;
- USB Female-to-Female connector type A;
- USB type A to micro-USB type B cable.

1.1 Preparing the host computer

This section sets up the necessary utilities on the computer, in order to build the Android framework and kernel.

1.1.1 Preparing Ubuntu 10.04

Install the necessary packages on the host platform, using the following command:

```
sudo apt-get install git-core gnupg flex bison gperf zip curl
gcc-multilib g++-multilib build-essential
```

1.1.2 Install Java SDK

Install Java Development Kit version 5.

1. Open the `source.list` file:

   ```
   sudo gedit /etc/apt/source.list
   ```

2. Add the `Jaunty` repositories to this file:

   ```
   deb http://us.archive.ubuntu.com/ubuntu/ jaunty multiverse
   deb http://us.archive.ubuntu.com/ubuntu/ jaunty-updates multiverse
   ```

3. Update the package list:

   ```
   sudo apt-get update
   ```

4. Install Sun Java 1.5:

   ```
   sudo apt-get install sun-java5-jdk
   ```

5. Check the default version of Java:

   ```
   java -version
   ```

6. If the output of `java -version` is not `Java version "1.5.0_*`, set Java 1.5.0_* to default using the following command:

   ```
   sudo update-java-alternatives -s java-1.5.0-sun
   ```

1.1.3 Installing Android SDK

In this chapter, the Android SDK will be installed:

1. Download the SDK from `http://developer.android.com/sdk/index.html`.
2. Pick the Linux version and unzip it in `~/android-sdk-linux_86`.
3. Start the `android` program: `~/android-sdk-linux_86/tools/android`.
4. Go to `Available Packages` in the left panel.
5. Unfold the `Android repository` menu. Select at least `Android SDK Platform-Tools, revision2` and `SDK Platform Android 2.2 API 8`.
6. Click on the `Install Selected` button.

1.1.4 Installing the Android NDK

The NDK cross compiler is installed in this chapter:

1. Download the NDK from `http://developer.android.com/sdk/ndk/index.html`.
2. Pick the Linux version and unzip it in `~/android-ndk-{release}-linux-x86`.

1.1.5 Exporting $PATH

For convenience, it is recommended to add NDK's cross compiler and the the SDK `platform-tools` folder to `$PATH`:

```
export PATH=$PATH:/home/user/android-sdk-{release}-linux_86/tools/:
/home/user/android-ndk-{release}-linux-x86/toolchains/
arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/
```

1.1.6 Creating Android Rulefile

Create a rulefile in Ubuntu. Add a rule to this file, which is applicable for HTC mobile phones (The Nexus One is manufactured by HTC):

1. Create a file in `/etc/udev/rules.d` named `50-android.rules`:

   ```
   sudo gedit /etc/udev/rules.d/50-android.rules
   ```

2. Add the following text:

   ```
   SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
   ```

3. Change file permission:

   ```
   sudo chmod a+rx /etc/udev/rules.d/50-android.rules
   ```

1.1.7 Downloading the Android Sources

The host host computer is now prepared for this project. The next step is to download the Android source code from the repository (which takes +/- 1.5 hours) and compile the source code (which takes another +/- 1.5 hours).

1. Download and install the `repo` client:

   ```
   mkdir ~/bin
   curl http://android.git.kernel.org/repo > ~/bin/repo
   chmod a+x ~/bin/repo
   export PATH=$PATH:~/bin
   ```

2. Synchronize with the Android source code repository:

   ```
   mkdir myandroid
   cd myandroid
   repo init -u git://android.git.kernel.org/platform/manifest.git \
   -b froyo-release
   repo sync
   ```

NOTE: It can happen that, during this process a `connection time-out` occurs. Reissue `repo sync`, the process will continue from where it stopped.

3. Everything should be downloaded on this point. However, one modification in the source code is required to gain `su` privileges later on. Therefore, open de `su.c` file and remove the user permission check:

   ```
   gedit ~/myandroid/system/extras/su/su.c
   ```

4. Comment line 62 - 65 with the following content:

   ```
   //    if (myuid != AID_ROOT && myuid != AID_SHELL) {
   //        fprintf(stderr,"su: uid %d not allowed to su\n", myuid);
   //        return 1;
   //    }
   ```

5. Enter the flowing commands to compile the Android Framework:

   ```
   cd ~/myandroid
   . build/envsetup.sh
   lunch full_passion-eng
   make -j4 CROSS_COMPILE=arm-linux-androideabi- ARCH=arm
   ```

1.2 Prepare the Nexus One

To modify the operating systems internals, it is required to obtain root privilege. This section explains how to obtain this privilege. It is important to note that this operation erases all data on the Device. First, the `ConnectBot` program is installed to use a console on the Nexus One

1.2.1 Install ConnectBot

Install `ConnectBot` on the Nexus One:

1. Enable `USB Debugging` on the device:

   `Applications menu -> Settings -> Applications -> Development`

2. Download the `ConnectBot` application, go to: `https://code.google.com/p/connectbot/`, download the `.apk` file, save this file in `~/Download`.

3. Install `ConnectBot.apk` on the Nexus One:

   ```
   adb install ~/Download/ConnectBot.apk
   ```

1.2.2 Unlock the Nexus One

1. Lookup the build number. This number can be found on the Android device:

   `Applications menu -> Settings -> About Phone -> Build Number`

2. Download the appropriate fastboot program. Select this the appropriate version by its build number, and place it in a directory (e.g. `~/fastboot/`).

   `http://android.modaco.com/content/google-nexus-one-nexusone-modaco-com/298782/23-nov-superboot-erd79-frg83d-rooting-the-nexus-one/`

3. Reboot the Nexus One into the bootloader menu:

   ```
   adb reboot-bootloader
   ```

4. When the Nexus One has been rebooted into the boot loader, the device can be unlocked using the `fastboot` program:

   ```
   ~/fastboot/fastboot-linux oem unlock
   ```

5. The instructions from the screen can be followed.

1.3 Add NFC and USB support to Android

This section explains how USB and NFC support is added to Android.

1.3.1 Add USB Library to Android

This section explains how to port the usblib to the Android platform. This part is adapted from `http://android.serverbox.ch/wp-content/uploads/2010/01/android_industrial_automation.pdf`.

1. Download `libusb-0.1.12` from `http://downloads.sourceforge.net/libusb/libusb-0.1.12.tar.gz` and extract it in `~/myandroid/external`.

2. Remove all unnecessary files, all files which do not have a `.h`, `.cpp` or `.c` extension.

3. Create `Android.mk` in `./external/libusb-0.1.12`, and add the following code:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES:= \
    error.c \
    linux.c \
    descriptors.c \
    usb.c

LOCAL_C_INCLUDES := external/libusb-0.1.12
LOCAL_PRELINK_MODULE := false
LOCAL_MODULE := libusb
include $(BUILD_SHARED_LIBRARY)
```

4. Add the usb library to the framework:

```
mmm ~/myandroid/external/libusb-0.1.12
```

1.3.2 Add NFC library

This chapter explains how the NFC library is ported to the Android environment.

1. Obtain `libnfc-1.4.1` from `https://code.google.com/p/libnfc/downloads/detail?name=libnfc-1.4.1.tar.gz` and unpack this file in: `~/myandroid/external/libnfc-1.4.1`.

2. As in `libnfc-1.4.1` (recursively) remove all files which do not have a `.c` or `.h` extension.

3. Add Android.mk in `~/myandroid/external/libnfc-0.1.12`:

```
LOCAL_PATH := $(call my-dir)
subdirs := $(addprefix $(LOCAL_PATH)/,$(addsuffix /Android.mk, \
libnfc ))
include $(subdirs)
```

4. Add Android.mk in `~/myandroid/external/libnfc-0.1.12/libnfc`:

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:= \
   buses/uart.c  \
   chips/pn53x.c \
   drivers/arygon.c \
   nfc.c \
   iso14443-subr.c \
   mirror-subr.c

LOCAL_CFLAGS := -O2 -g -std=c99 -DSERIAL_AUTOPROBE_ENABLED
LOCAL_CFLAGS += -DHAVE_CONFIG_H -DHAVE_LIBUSB -DDRIVER_ARYGON_ENABLED
```

```
LOCAL_C_INCLUDES += \
    external/libnfc/ \
    external/libnfc/include/ \
    external/libnfc/libnfc/buses/ \
    external/libnfc/libnfc/chips/

LOCAL_SHARED_LIBRARIES += libusb

LOCAL_MODULE:= libnfc
LOCAL_PRELINK_MODULE:= false

include $(BUILD_SHARED_LIBRARY)
```

5. Add a `MIN()` and `MAX()` macro in `~/myandroid/external/libnfc-0.1.12/include/nfc/nfc.h`

```
#define MIN(a, b) ((a) < (b) ? (a) : (b))
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

6. Add "`#include nfc/nfc.h`" to:
   `./libnfc-1.4.1/libnfc/drivers/arygon.c`
   `./libnfc-1.4.1/libnfc/buses/uart_posix.c`
   `./libnfc-1.4.1/libnfc/chips/pn53x.c`

7. Add the NFC library to the framework:

```
mmm ~/myandroid/external/libnfc-1.4.1
```

1.3.3 Add nfc-list

To test whether the `libnfc` implementation works, `nfc-list.c` is compiled in the same way as `lsusb`.

1. Create a new directory `nfc-list` in `~/myandroid/external/nfc-list`. Copy the `nfc-list.c`, `nfc-utils.c` and `nfc-utils.h` to this directory:

```
mkdir ~/myandroid/external/nfc-list
cp ~/myandroid/external/libusb-1.4.1/example/list-nfc.c \
   ~/myandroid/external/nfc-list/
cp ~/myandroid/external/libusb-1.4.1/example/nfc-utils.* \
   ~/myandroid/external/nfc-list/
```

2. Create `Android.mk` in `~/myandroid/external/nfc-list` to compile the source files:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES     :=  \
    nfc-list.c
    nfc-utils.c

LOCAL_MODULE        := nfc-list
LOCAL_C_INCLUDES    += \
    external/libnfc-1.4.1/ \
    external/libnfc-1.4.1/include/

LOCAL_SHARED_LIBRARIES := libc libusb libnfc
include $(BUILD_EXECUTABLE)
```

3. Add `nfc-list` to the framework:

```
mmm ~/myandroid/external/nfc-list
```

1.4 Android Kernel and Kernel Modules

This chapter is a guide to obtain and compile the Android kernel source. The standard source code is in this case not suitable. The goal of the new kernel is to let the Android device act like a USB-host, this is not the case by the default source code. However, a German developer named Sven Killig `http://sven.killig.de/android/N1/2.2/usb_host/` has developed and released a kernel source which let the Android device work like a USB-host.

1.4.1 Compile Android Kernel

- Obtain the modified Android kernel sources:

```
cd ~/
git clone http://github.com/sonic74/kernel_msm.git
```

- Download the `config.gz` file from the device and unzip it:

```
cd kernel_msm
adb pull /proc/config.gz
gunzip config.gz
mv config .config
```

- Compile kernel sources:

```
make -j4 ARCH=arm CROSS_COMPILE=arm-linux-androideabi-
```

1.4.2 Obtaining Kernel Modules

To install the `Connectbot` application. This application is a terminal, this terminal is needed, since no USB connection to the Nexus One is possible after booting the new image (the Nexus One is then a USB-host). Several kernel modules need to be downloaded first. These modules load the USB and NFC driver.

- Download these kernel modules from Sven Killig's website:

```
usbcore.ko
ehci-hcd.ko
usbserial.ko
cp210x.ko
```

- Copy these modules to the sdcard, placed inside the device.

```
adb push usbcore.ko /sdcard/usbcore.ko
adb push ehci-hcd.ko /sdcard/ehci-hcd.ko
adb push usbserial.ko /sdcard/usbserial.ko
adb push cp210x.ko /sdcardcp210x.ko
```

1.5 Putting it all together

By using this appendix, the next things should be ready:

- Ubuntu is ready;
- SDK and SDK are installed;
- The Android Framework is compiled;
- The Android kernel is compiled;
- The kernel modules are loaded on the SDcard;
- The `ConnectBot` application is downloaded;
- The Nexus One is unlocked;
- The Fastboot program is available on the system.

The next step is, to load the new compiled kernel and the framework on the Nexus One. The fist step is to place the kernel into the boot image. Second, flash system image and boot image to the Nexus One.

1. Make a directory for the boot image. This image needs to be modified:

   ```
   mkdir ~/images
   ```

2. Copy the system and kernel image from `~/myandroid/out/target/out/target/product/passion/` to `~/images`:

   ```
   cp \
   ~/myandroid/out/target/out/target/product/passion/boot.img \
   ~/images/
   cp \
   ~/myandroid/out/target/out/target/product/passion/system.img \
   ~/images/
   ```

3. The `boot.img` needs to be unpacked, in order to replace the kernel with the kernel build in Section 1.4.1. Therefore, download the unpack and repack script from respectively: `http://android-dls.com/files/apps/unpack-bootimg.zip` and `http://android-dls.com/files/apps/repack-bootimg.zip`. Unzip these scripts and place them in /script. Modify line number 19 from `repack-bootimg.pl`: add `--base0x20000000` right after `mkbootimg`.

```
cd ~/images
~/script/unpack-bootimg.pl boot.img
~/script/repack-bootimg.pl ~/kernel_msm/arch/arm/boot/zImage \
boot.img-ramdisk new-image.img
```

1.5.1 Flashing the Nexus One

4. Restart the nexus one into the bootloader:

```
adb reboot-bootloader
```

5. Flash the system partition:

```
../fastboot/fasboot-linux flash system ~/images/system.img
```

6. Load the kernel, do not flash it.

```
../fastboot/fasboot-linux boot ~/images/new-image.img
```

7. This automatically boots the device with the new kernel and new system image.

1.5.2 Last step, make it work

When the Nexus is successfully started, make sure all cables are connected in the proper way.

1. Start `ConnectBot` to load the kernel modules:

```
su
cd /sdcard
insmod usbcore.ko
insmod ehci-hcd.ko
mount -t usbfs none /proc/bus/usb
insmod usbserial.ko
insmod cp210x.ko
```

2. To test if everything works, execute `nfc-list`.