Standard Operating Procedures for Android Embedded Systems

Anupama M. Kulkarni, Shang-Yang Chang, Ying-Dar Lin
National Chiao Tung University, Hsinchu, Taiwan

November 2012

Android is considered to be more of a software stack rather than just an operating system for mobile handsets [1]. The main reasons being its open source nature and the presence of application framework on top of Linux kernel. Unlike android application developers, system level engineers have limited online documentation. Usually, new engineers take a lot of time to get familiar with the procedures to build android embedded systems. Therefore, to save costs and reduce complexity, we suggest a set of standard operating procedures (SOP) for constructing a complete android-embedded system from scratch as shown in Figure 1 and test its operations on an android phone.

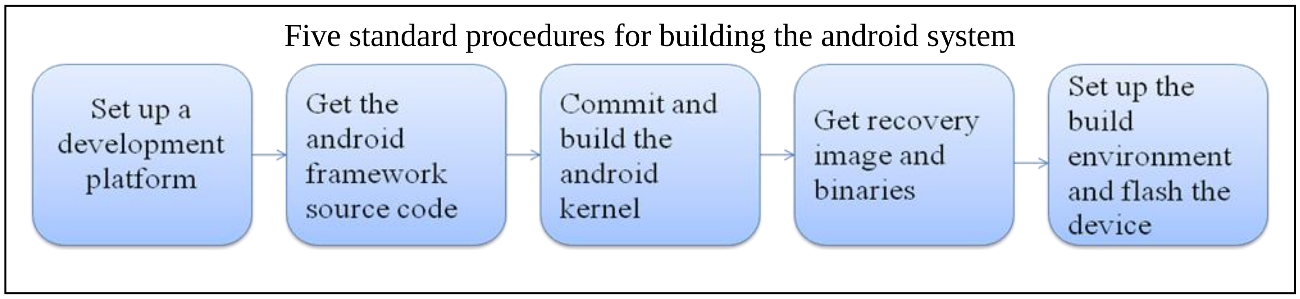
Five standard procedures for building the android system
Set up a development platform
Get the android framework source code
Commit and build the android kernel
Get recovery image and binaries
Set up the build environment and flash the device

Figure1. Procedures for building android embedded systems

Standard procedures for building android-systems

In order to build the Android tree, first step is to set up a development platform. Linux or Mac OS needs to be used as development platform. Usually, recent version of Ubuntu (6.06 and above) is preferred. If you are running Linux in a virtual machine, you will need at least 1.5GB of RAM and 10GB or more of disk space. Since Ubuntu doesn't come with all the necessary packages, we should make sure we have the following packages installed:

Table 1. Packages required for setting up the development platform and their importance

Packages	Necessity
1) Git (1.5.4 version or newer)	To access the online repositories.
2) Java Development Kit (JDK 5.0 or 6.0)	To compile Java programs.
3) Valgrind	A tool that to helps you find memory leaks, stack corruption, array bounds overflows, etc.
4) flex, bison, gnupg, gperf, libsdl-dev, libesd0-dev,	Essentials to compile and build various

libwxgtk2.6-dev (optional), build-essential, zip, curl, libncurses5-dev,zlib1g-dev.	files.

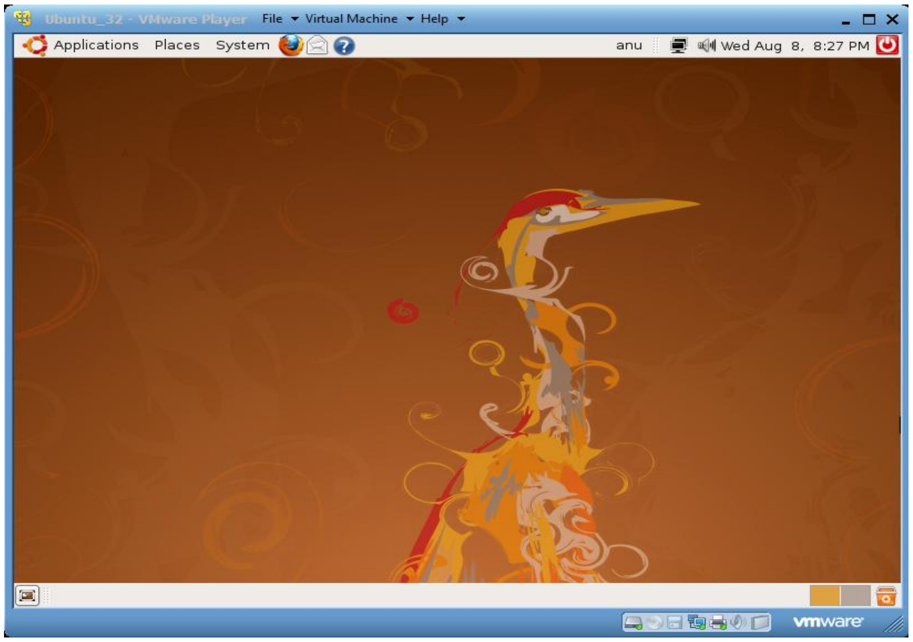

Figure 2. Linux (Ubuntu 8.04) development environment

Once we have set up the Linux environment as shown in Figure 2, and got the necessary packages, we proceed further with the second step of getting the android framework. It consists of the following [2]:

- Core Projects: Form the foundation of the Android platform. They include
    - Bionic, Bootloader, Dalvik, Frameworks, Hardware, Prebuilt, System.
- Externals: Other open source projects like Eclipse, apache , Flex, freetype, etc.
- Packages: Include user applications Alarm, Calculator, Calendar, Camera, Contacts, Browser.

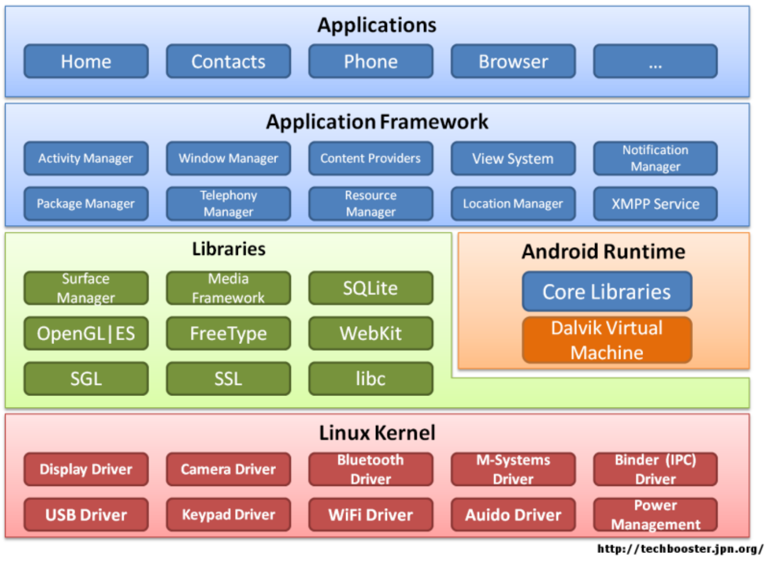

Figure 3. Android framework

The android framework shown in Figure 3, can be downloaded using "repo", which is a repository management tool. It unifies many Git repositories when necessary, does the uploads to our revision control system, and automates parts of the Android development workflow.[3] So, first download the repo script and make sure it is executable. Once you are equipped with repo, you can connect to the online android-repository by initializing a local client. Finally, SYNC and pull down the android source files from public depot to local directory.

Next, you need to get the kernel required for your target device. It contains executable files (.exe files) which are used to run every event occurring in the system. The source files are available on the internet. We need to clone or get a copy of them to our working directory. Next step will be to configure the kernel. This process involves specifying what functionality to put into the kernel. When building for another processor architecture than the one that your host processor uses you must use a crosscompiler and set the architecture parameter. For example if the target architecture is ARM, set the ARCH parameter to arm. To configure a kernel, you can either obtain the kernel configuration (config) file from the internet or a phone running android or get the default config options for the kernel from what is already specified in kernel source file using **defconfig**. Next, to build the kernel we set the CROSS_COMPILE environment variable. It stores the path to the arm cross compiling toolchain, (prebuilt) which comes with android source code. Finally, executing **make** will start the build process and builds the kernel according to the configuration files. The image is generated in the folder: arch/arm/boot/zImage.

Apart from the kernel image (binaries) and the system framework, we would also require the system recovery image and the proprietary binaries for the target device. So, in this step we simply download the required files from the target-device developer site and execute them from the working directory.

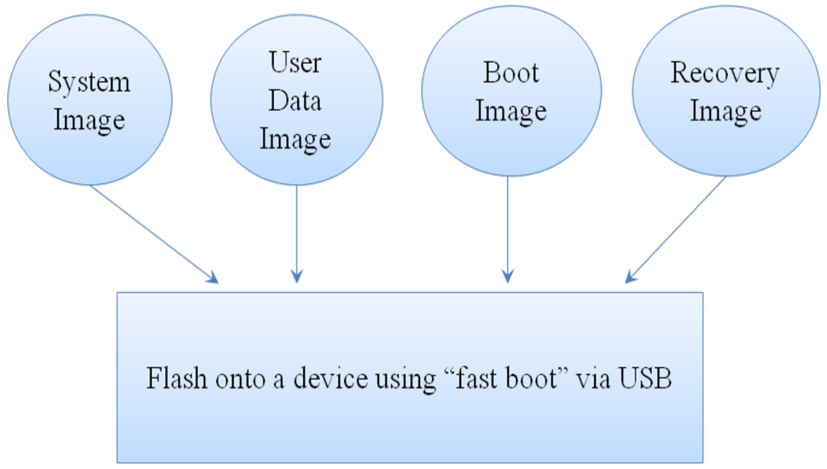

Figure 4. Four images that are flashed onto the target device

At this stage of the build process, you are having all the required files to build a complete embedded system, as shown in Figure 4: the **system image** that contains all the required binaries for the system, the **userdata image** that contains user specific and application specific data, the **boot image** which is responsible for the boot process of the device (includes **kernel** and **RAMdisk** image) and the **recovery image** which is required to perform the android recovery and maintenance operations. Images contain a copy of data present inside the components of the system. (For ex: A system image contains the state of the entire computer system stored in it). In the final step, we set up the necessary build environment and flash the images onto the target device. The emulator build environment can be set up either by including the buildspec.mk (Make file describing the build specifications) into the root directory or using the envsetup.sh script and lunch command. Finally, the target device is flashed using "fastboot" tool. It writes all the images together into the device. For doing this, we first get the device into fastboot mode. Using the appropriate fastboot commands, you can write all the images onto your target device.

By rebooting the device, you will be able to run it on the new kernel/system you just burnt. Proper functioning of various modules like dialpad, bluetooth, wi-fi, camera, etc., can be checked on the target device itself. For example, you can find the calculator and check if it is working well with some operations performed on it.

Conclusion

The present article describes a smooth pathway for building an android-based embedded system from scratch. We have tested these steps on a android 1.6 version phone. Also, these steps help you to understand the complete procedure and problems in implementing android-embedded systems.

References

1) ebook-O'Reilly-Karim Yaghmour, "Embedded android"
2) Android project layout available at: https://sites.google.com/a/android.com/opensource/projects
3) Version control with Repo and Git available at: http://source.android.com/source/version-control.html

4) Android basics available at :
http://developer.android.com/training/basics/firstapp/index.html
5) Android open source project available at :
http://developer.android.com/training/basics/firstapp/index.html
6) Fastboot available at : http://wiki.cyanogenmod.com/wiki/Fastboot
7) Collin walls .ppt, "Getting started with android development for embedded systems".