The first impression of Android source code

2008.11.10

京都マイクロコンピュータ
小林 哲之

はじめに

- 2008年10月22日についにAndroidのソースコードが公開されました。
- このソースをざっと見渡した印象について皆さんと情報共有しましょう。
- 内容は発表者個人の主観に基づいています。
- 正確性については「無保障」です。☺
- 再利用はご自由に。
- 10/30版から少し更新しました。

Who am I?

- 組み込み一筋N十年。
  - リアルタイムOS iTRON
  - 組み込み向けJava実行環境
  - 組み込み向けLinux
  - gcc
- ブログ「組み込みの人。」
  - http://d.hatena.ne.jp/embedded/
- 京都マイクロコンピュータ 2008年3月入社
  - http://www.kmckk.co.jp/

本日お話すること

- ソースの概要について
  - 内容
  - ライセンス
  - 移植性の検討
- 個別のトピックス
  - bionic
  - Dalvik VM
  - Linux kernel

ソースのある場所

- ここ
  - http://source.android.com/
- gitとそのラッパースクリプト(repo)を使用してソースをダウンロードする。
- ブラウザで閲覧するならここ
  - http://git.source.android.com/

何が含まれている？

- Android SDKを再ビルドに必要なものが全て。
- Linux カーネル、デバイスドライバ、標準Cライブラリ、コンパイラツールチェイン、Dalvik VM、2D/3Dグラフィックライブラリ、コーデック、アプリケーションフレームワーク、webkit(WEBブラウザ)、....
- ただし、MAPアプリケーションは含まれていない。

- HTC G1のソース？ (デフォルトではダウンロードされないがリポジトリに存在する。けっこう頻繁に更新。)
  - kernel/msm.git
  - platform/hardware/msm7k.git
  - platform/vender/htc/dream.git

全てのソースがあるのか？

- コンパイラツールチェインと一部のライブラリはバイナリで入っている。
  - $(TOP)/prebuilt 以下
- それらをリビルドするために必要なソースのありかが明記されている。
  - 各ディレクトリのREBUILTというファイルに

全部で何行？

```
$ find . ! –path ‘*/.git/*’ -type f |xargs cat | wc
22998555 81705021 996249064
$
```

つまり2300万行

Linuxカーネルが1000万行
external(外部ライブラリ、ツール)で800万行
残りが500万行。
- frameworks 100万行
- dalvikvm 100万行

ライセンス条件は？

- 基本はApache 2.0
- 多数のオープンソースの成果を利用しているので、それはそれぞれのライセンス条件
- モジュールごとに MODULE_LICENSE_*** というファイルがあり、簡単に確認できるように整備されている。
- また、それぞれライセンス表示のための NOTICEというファイルがおかれている。

```
$ find . -name "MODULE_LICENSE_*“
./system/extras/showmap/MODULE_LICENSE_APACHE2
./system/extras/showslab/MODULE_LICENSE_APACHE2
./system/extras/librank/MODULE_LICENSE_APACHE2
./system/extras/procrank/MODULE_LICENSE_APACHE2
./system/extras/latencytop/MODULE_LICENSE_APACHE2
    ...
```

```
$ find . -name "MODULE_LICENSE_*" |sed 's,^.*/,,
' |sort |uniq -c |sort -r
    117 MODULE_LICENSE_APACHE2
     24 MODULE_LICENSE_BSD_LIKE
     11 MODULE_LICENSE_BSD
     10 MODULE_LICENSE_GPL
      9 MODULE_LICENSE_EPL
      7 MODULE_LICENSE_LGPL
      4 MODULE_LICENSE_PUBLIC_DOMAIN
      4 MODULE_LICENSE_LGPL_AND_GPL
      2 MODULE_LICENSE_CPL
      1 MODULE_LICENSE_W3C
      1 MODULE_LICENSE_OSL1
      1 MODULE_LICENSE_MIT
      1 MODULE_LICENSE_HP
      1 MODULE_LICENSE_AFL_AND_GPL
```

ちなみにGPLのものは

```
$ find . -name "MODULE_LICENSE_*" |grep GPL
./external/iptables/MODULE_LICENSE_GPL
./external/elfcopy/MODULE_LICENSE_GPL
./external/jdiff/MODULE_LICENSE_LGPL
./external/yaffs2/MODULE_LICENSE_GPL
./external/qemu/MODULE_LICENSE_GPL
./external/webkit/WebCore/MODULE_LICENSE_LGPL
./external/dbus/MODULE_LICENSE_AFL_AND_GPL
./external/oprofile/MODULE_LICENSE_GPL
./prebuilt/windows/sdl/MODULE_LICENSE_LGPL
./prebuilt/windows/ccache/MODULE_LICENSE_GPL
./prebuilt/darwin-x86/sdl/MODULE_LICENSE_LGPL
./prebuilt/darwin-x86/toolchain/i686-apple-darwin8-
4.0.1/MODULE_LICENSE_LGPL_AND_GPL
./prebuilt/darwin-x86/toolchain/arm-eabi-4.2.1/MODULE_LICENSE_LGPL_AND_GPL
./prebuilt/darwin-x86/ccache/MODULE_LICENSE_GPL
./prebuilt/darwin-x86/make/MODULE_LICENSE_GPL
./prebuilt/common/jfreechart/MODULE_LICENSE_LGPL
./prebuilt/common/swing-worker/MODULE_LICENSE_LGPL
./prebuilt/common/netbeans-visual/MODULE_LICENSE_GPL
./prebuilt/linux-x86/sdl/MODULE_LICENSE_LGPL
./prebuilt/linux-x86/toolchain/i686-linux-gnu-
3.4.6/MODULE_LICENSE_LGPL_AND_GPL
./prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/MODULE_LICENSE_LGPL_AND_GPL
./prebuilt/linux-x86/ccache/MODULE_LICENSE_GPL
```

ビルドのしかた

- Linux (ubuntu), MacOS(intel)でのビルドがサポートされている。
  - http://source.android.com/download
  - 他に必要なライブラリ
    - sudo apt-get install zlib1g-dev libncruses5-dev unzip

- 手軽な方法
  - VMWare 上にubuntu 8.04 i386をクリーンインストール
  - 無料のVMWare player利用可能
  - ビルド時間 40分くらい？
- 困った時はネットで検索。☺
- make sdk でSDK一式が再ビルド。

移植性？

- ターゲットCPUは現在はARMのみ。
  - armv5t (ARM926以降、Xscale)
- パス名に'arm'を含むファイルが多いもの
  - bionic
  - dalvik
  - external/opencore
  - external/sonivox
  - external/qemu
  - external/openssl
- これらの移植に目途がつけば、他のCPUでもいけそう。

移植性?: Bionic

- 標準ライブラリとランタイムローダ
  - libc, libm, libdl, libthread_db, linker
- FreeBSD, OpenBSD, NetBSDからのファイルをベースにしているようだ。
- アセンブラで書いてあるのは
  - カーネルのシステムコール呼び出し部分
  - memcpyなどのアセンブラ版
- 現在はARM用のものしかないが、同じように*BSDから持ってくれば他のCPU用のものも準備できそう。

移植性?: Dalvik VM

- DXコードのインタプリタが何種類か用意されている。
  - 高速版 (ARMのアセンブラで書かれている)
  - 移植用 (Cで書かれている）
  - デバッグ用、プロファイル用
- それ以外のARM依存ファイルはABIを定義しているファイルのみ。
- とりあえずC版インタプリタを使えば移植は簡単？
- コンフィグの設定だけでx86向けにはビルドできるそうだ。

互換性テスト

- 各モジュールに単体テスト用のファイルらしきものがあるが、システム全体に渡る互換性はどうなるのだろう…？
- キラーアプリ、キラーデバイスによるデファクトスタンダード？
- サブセット化OK?

ここまでの感想

- 「本当に」全部のソースが公開された。
- 扱いやすいライセンス。
- 用途は携帯電話に限定されない。
- まさに、Google 太っ腹！ここから直接収益を上げることは全く考えていないようだ。「どうぞ、使ってください」という感じ。
- 組み込みLinuxのメジャーなディストリビューションになるかも。

- 個別のトピックス

Bionic

- CAVEAT(制約事項)
  - $(TOP)/bionic/libc/CAVEATS
  - 基本はPOSIX準拠だが
  - C++の例外をサポートしない。そのためSTL(標準テンプレートライブラリ)もない。
    - Vector.h, List.h はframeworks/base/include/utils にある。
  - pthreadのキャンセル機能はサポートしない。
  - ロケールやワイドキャラクタはサポートしない。国際化対応には代わりにICUを使う。

libstdc++ の中身はたったこれだけ。

```
$ nm -C out/target/product/generic/symbols/system/lib/libstdc++.so |grep ' [TDRB] '
000019dd T type_info::type_info(type_info const&)
000019a5 T type_info::type_info()
000019dd T type_info::type_info(type_info const&)
000019a5 T type_info::type_info()
000019f5 T type_info::~type_info()
00001a19 T type_info::~type_info()
00001a19 T type_info::~type_info()
000019bd T type_info::name() const
000019d9 T type_info::before(type_info const&) const
000019d1 T type_info::operator==(type_info const&) const
000019d5 T type_info::operator!=(type_info const&) const
00001a30 R std::nothrow
00002000 D vtable for type_info
0000192d T operator delete[](void*)
0000190d T operator delete[](void*, std::nothrow_t const&)
0000193d T operator delete(void*)
0000191d T operator delete(void*, std::nothrow_t const&)
00001965 T operator new[](unsigned int)
0000194d T operator new[](unsigned int, std::nothrow_t const&)
00001971 T operator new(unsigned int)
00001959 T operator new(unsigned int, std::nothrow_t const&)
0000189d T __cxa_guard_abort
000018d9 T __cxa_guard_acquire
000018b5 T __cxa_guard_release
0000197d T __cxa_pure_virtual
$
```

ダイナミックリンク

- linker
  - ld.so ではない。Android 新規設計。
  - libdlの機能はある。dlsym, dlopen
- prelink (apriori)
  - ライブラリを固定アドレスにマッピング
    - build/core/prelink-linux-arm.map
  - 実行ファイル(executable)はprelinkしてない
  - 起動高速化というより共有メモリの効率化のため?
- stripコマンドの代わりにsoslimコマンド

各ライブラリの配置アドレス

```
$ cat build/core/prelink-linux-arm.map

# 0xC0000000 - 0xFFFFFFFF Kernel
# 0xB0100000 - 0xBFFFFFFF Thread 0 Stack
# 0xB0000000 - 0xB00FFFFF Linker
# 0xA0000000 - 0xBFFFFFFF Prelinked System Libraries
# 0x90000000 - 0x9FFFFFFF Prelinked App Libraries
# 0x80000000 - 0x8FFFFFFF Non-prelinked Libraries
# 0x40000000 - 0x7FFFFFFF mmap'd stuff
# 0x10000000 - 0x3FFFFFFF Thread Stacks
# 0x00000000 - 0x0FFFFFFF .text / .data / heap

# core system libraries
libdl.so                0xAFF00000
libc.so                 0xAFE00000
libstdc++.so            0xAFD00000
libm.so                 0xAFC00000
liblog.so               0xAFBC0000
libcutils.so            0xAFB00000
libthread_db.so         0xAFA00000
libz.so                 0xAF900000
libevent.so             0xAF800000
libssl.so               0xAF700000
libcrypto.so            0xAF500000

   ....
```

Dalvik

- クラスライブラリはHarmonyプロジェクトの成果、CLASSPATHだとGPL
- アプリ起動時、zygoteというプロセスからforkされ、新しいプロセスのmainメソッドをDalvik VMのレベルでインボークされる。exec システムコールは使用していない。
  - なるべく多くの共有メモリをzygoteから引き継ぐ。
  - アプリ起動時間の短縮にもなる。

DXコードインタプリタ

- JIT(Just In Time compiler)は装備していない。
- 複数のインタープリタ実装
  - 高速版 (ARMのアセンブラで書いてある。)
  - デバッグ、プロファイル用
  - 移植用 (C言語で書いてある。)
- ARMアセンブラではFPU命令が使われていない。(ARM1136jf向けにチューンする余地)

Linux Kernel

- 2.6.25
- Android特有の部分
  - binder
  - ashmem
- ユーザー空間デバイスドライバ？
  - libhardware

binder

- プロセス間通信
- OpenBinder.org のものをベースにしていたが今は互換性はない。
- Androidのアプリケーションフレームワークの根幹をなす。
- 上位層は AIDL につながっている。

ashmem

- Android / Anonymous SHared MEMory subsystem
- $(TOP)/system/core/cutils/ashmem.h
  - int ashmem_create_region(const char *name, size_t size)　→ returns fd
  - int ashmem_set_prot_region(int fd, int prot)
  - int ashmem_pin_region(int fd, size_t offset, size_t len)
  - int ashmem_unpin_region(int fd, size_t offset, size_t len)
- ‘pin’ していないメモリはカーネルが回収して再利用
- Javaの weak reference に似ている概念。キャッシュの実装に便利。
- Javaから利用するときはandroid.os.MemoryFile

ビルド

- make showcommands
- ccache を使うためには環境変数 USE_CCACHE=1
- VMWare環境ではデフォルト設定が無難 (512MBメモリ、プロセッサ1個)
- make sdk
  - JDK6でなくJDK5でないとjavadocでエラーになる

最後に感想

- Androidのソースは宝の山！
- サーバやPCのLinux環境を削って作ったものではなく、既存の常識を払拭して、小さなシステムにフォーカスした設計になっている。
- Linuxの仮想メモリの機能を十二分に活用しようとしている。
- 新規プロジェクトのプラットフォームの候補としてとても面白い。ぜひコミュニティで情報交換を！