携帯電話（スマートフォン）の仕組み～Android の中の Linux

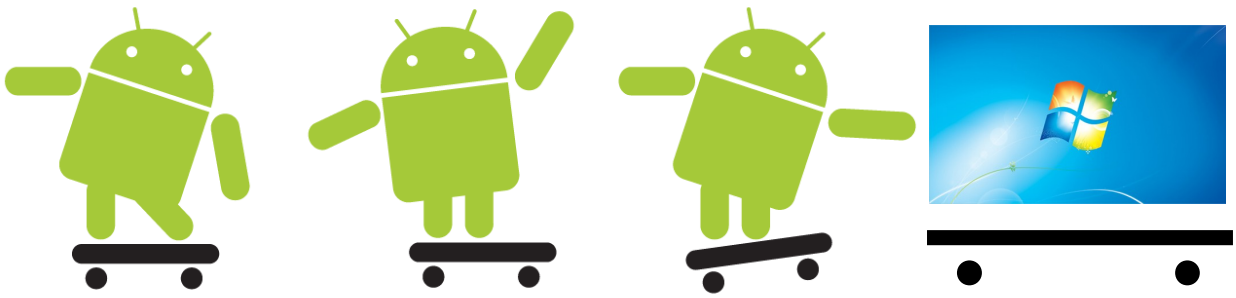
1. はじめに

今や携帯電話におけるスマートフォンの勢力は絶大なものとなっていることは言うまでもありません。スマートフォンは、その使用される OS に、アップル社が提供する iPhone、グーグル社が提供する Android、マイクロソフト社が提供する Windows Phone といった主要なOSがあります。

今回はAndroid携帯のOSについて、その構造について見ていきます。
なお、Sharp製　AU スマートフォン　IS05 を見ていますが、基本的には他の機種も同等の仕組みになっていますので参考になると思います。

そして、主にそのファイルシステムの構造をメインに見ていきますが、そのため必要となる、Android 開発環境についても触れていきます。

また、携帯自体がLinuxであることから、内部を調べるツールも、Linuxで実行するツールしか出回っていないような場合もありますが、今回は、PCは **Windows のみ**を使用した環境を構築するようにしています。



2. Android　OSとは

Android　OSとは、先ほど述べたように、グーグル社が提供するスマートフォン用プラットフォームのことですが、そのベースはLinuxであることはご存知の人も多いと思います。

つまり、Linux　OSの上にAndroid　OSが存在して、そのAndroid　OSの上に携帯電話のさまざまな機能やアプリケーションが乗って動作を行うことになります。

Android　OSは、仮想マシン（Dalvic VM）を持ち、Javaアプリケーションを動作させることができるようになっています。多くの公開されているアプリケーションなども、Javaアプリケーションで作られています。

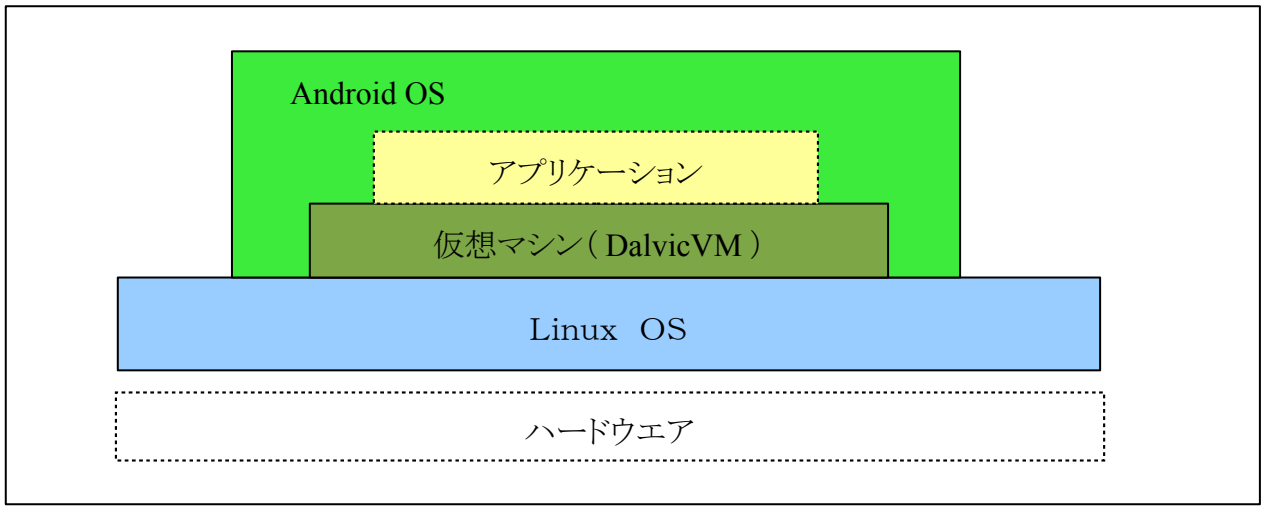


図1　システム概略

Androidアプリケーションを作成するためには、JavaSDKとEclips（デバッグ環境）、AndroidSDKをパソコンにインストールすることで、Javaベースのアプリケーションを作成することができます。

今回は、アプリを作ることが目的ではないので、 Android上のアプリケーションの開発の HowTo については別途上記関連サイトや専門書を参考にしていただきたいと思います。

参考）技術評論社　世界を目指せ！Androidアプリ開発入門　http://gihyo.jp/dev/serial/01/androidapp

また、Java ではなくネイティブな実行コードを作成する環境も提供されており、Android　NDK を利用することで、アプリをさらに高速に動作させることもできるようになっています。

なおAndroidSDKに含まれる、adbツールを用いると、携帯電話をUSBを経由して、直接アプリをインストールしたり、携帯のシステムをアクセスすることができるので、本文でもこのadbを利用した操作等を紹介しています。

また携帯の Linux シェル上で動作するコマンドを作成するのに Android　NDK を利用できます。

そのため Android 開発環境のインストールについて述べておきます。

3. 開発環境について

開発環境は、Windows だけでできる開発について、述べておきます。
OS は windows7(64bit)で述べますが、それぞれの環境に読み替えてください

・Android アプリ（apk）の開発環境（java プログラム）

1）JavaJDK のインストール＜執筆時のバージョン JDK 7＞ 64bit 使用 (Windows 用)
http://www.oracle.com/technetwork/java/javase/downloads/index.html
からインストールプログラムをダウンロードしてインストールします。

2）AndroidSDK のインストール＜執筆時のバージョン r15＞ (Windows 用)
http://developer.android.com/sdk/index.html
から zip ファイルのほうを取得して任意のフォルダに圧縮展開します。（インストーラ版もある）
Windows の環境（スタートー＞コンピュータ（右クリック）ー＞プロパティー>システム詳細設定
ー＞環境変数）にて Path の追加を行います。
tools へのパス C:\（android-sdk フォルダ）\tools\;
adb へのパス C:\（android-sdk フォルダ）\platform-tools\

3）Ecripse（開発間環境ツール）＜執筆時のバージョン 3.7.1＞ 64bit 使用(Windows 用)
http://www.eclipse.org/downloads/
から、Eclips IDE for Java Developers をダウンロードし、任意のフォルダに圧縮展開します
Eclipse（eclipse.exe）を起動するためのショートカットをデスクトップに作成しておくと便利です。
Eclipse を起動し、初回、ワークエリア、初回 AndroidSDK パス等聞かれた場合には指示に従います。
ープラグインの設定
[Help]->[Install New Software]で Install ダイアログが表示されたら［Add］をクリックして
Name：Android Plugin
Location：https://dl-ssl.google.com/android/eclipse/
を登録後、
［Work with］=>「Android Plugin」を選択します。
一覧が出ると、［Developer Tools］をチェックして NEXT＞をクリックしてインストールを開始します。
完了すると、RESTART の確認を実行します
ーAndroidSDK の設定
［Window］メニューの［Preferences］を開き、左の一覧から、”Android”を選択し、［SDK Location］に
Android SDK のパスを設定します。 C:\（android-sdk フォルダ）\
ーエミュレータの起動設定
エミュレータで動作確認する場合そのターゲットを作ります
［Window］メニューの［AVD Manager］->［New］を開き、
［Name:］適当な名前
［Target:］Android xxx（実機にあわせて）
［SD Card:］利用する擬似 SD カード領域 例 32M
［Skin:］画面解像度を設定
を設定して［Create AVD］で作成、作成した項目を選んで[Start]でエミュレータが立ち上がります
以上で AndroidSDK による開発環境のインストールは完了です。

Eclipse でプログラムのビルドからエミュレータの実行、デバッグができるようになっています。
SDK でのアプリの作成については、関連書籍やサイトを参照にしてください。

・Android ネイティブ実行ファイルの開発環境

Windows 上で Java ではなく、C 言語によるネイティブコードの生成に必要な環境です。
NDK には擬似 Linux 環境である sygwin が必要です。

1）Cygwin のインストール ＜執筆時のバージョン 1.7.9-1.＞
http://www.cygwin.com/
から setup.exe を実行しインターネットからダウンロード＆インストールします。
なおｾｯﾄｱｯﾌﾟは追加モジュールをインストールしたりするのに必要なので、任意のフォルダに入れて、
実行できるようにしておき、同フォルダ内にダウンロード一時ファイルを格納するようにしておきます。
インストールの際に、モジュールの選択で、
Devel ＞ gcc compiler upgrade Helper
Devel ＞ The GNU version of the 'make' utility
は、必要となるのでチェックを入れること。（NDK で最低必要なモジュール）
なお Setup を実行することで後から追加モジュールの追加ができます
Cygwin を起動すると以下のような、Linux 同等のコマンドプロンプトが表示されます。

```
$
```

2） Android NDK のインストール ＜執筆時のバージョン r7＞
http://developer.android.com/sdk/ndk/index.html
からダウンロードした圧縮ファイルを展開して、Cygwin 配下のフォルダに置きます。
(例\sygwin\home\(user)\android-ndk)
（Zip 展開すると android-ndk 以下が読み取り専用になっていたので、読み書き可能に変更してます）
そして Cygwin で AndroidNDK のビルドを行うため初期パスを追加しておきます。これには、
\sygwin\home\(user)\.bashrc ファイルの最後に
export ANDROID_NDK_ROOT=C:/cygwin/home/(user)/android-ndk
export PATH=$PATH:/cygdrive/c/cygwin/home/(user)/android-ndk
を追加しておきます。

3）不具合の対応
実はこれだけでは不具合があることがわかっています。そこで Cygwin を起動して

```
$which awk
  /usr/bin/awk
$which gawk
  /usr/bin/gawk
```

で awk、gawk があることを確認して、
C:\（cygwin フォルダ）\home\(user)\android-ndk\prebuilt\windows\bin にある、
awk.exe を awk_exe に名前を変更しておきます。
NDK の awk.exe だとエラーが出るので、/user/bin のを利用するためです

以上で NDK のインストールは完了です。

☆NDK のサンプルコードのビルドは、Cygwin を起動して AndroidManifest.xml のあるフォルダに移動し、
$ndk-build
を実行するだけです。
NDK は基本的に、Android Java アプリ(apk)から呼び出し可能なライブラリファイル（xxxxx.so）を作成することを目的としています。作成したライブラリを AndroidSDK のプロジェクトにいれてアプリを作ります

実際は、サンプル配下にあるディレクトリの構造がプロジェクトのディレクトリ構造となっており、
jni フォルダ（C ソースと Amdroid.mk ファイルがここに入ります）
res フォルダ
src フォルダ
tests フォルダ
AndroidManifest.xml
default.properties
のような構成になっており、このプロジェクトフォルダは、/home/(user)/の任意のフォルダにあってもよい

☆NDK で実行可能なファイルを作成することもできます。
ソースファイルと Make（Android.mk）ファイルを jni フォルダの配下にいれて、
Android.mk に include $(BUILD_EXECUTABLE)を記載します。

例：hello-exe
<hello-exe.c>
```
#include <stdio.h>
int main(int argc, char ** argv)
{
    printf("Hello, world!\n");
    return 0;
}
```
<Android.mk>
```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := hello-exe
LOCAL_SRC_FILES := hello-exe.c
include $(BUILD_EXECUTABLE)
```

として
home/(user)/hello-exe/jni/ の下に hello-exe.c と Android.mk を置いて Cygwin を立ち上げ、
$cd /home/(user)/hello-exe
$ndk-build
を実行するだけである。lib フォルダに実行可能なファイルが生成されます。
なおソースファイルが複数ある場合は、
```
LOCAL_SRC_FILES := ￥
                   source1.c￥
                   source2.c￥
```
と記述します。

・ADB 接続用 USB ドライバーのインストール
Sharp 製スマートフォンの場合、 https://sh-dev.sharp.co.jp/android/modules/driver/ から
Sharp 共通 ADB USB ドライバをダウンロードしてインストールします。
（各メーカーサイトや携帯キャリアサイトで確認してください）

以上の環境が Windows で Android アプリを開発する環境となります。
今回、仕組みを調べる際に必要に応じて使用します。

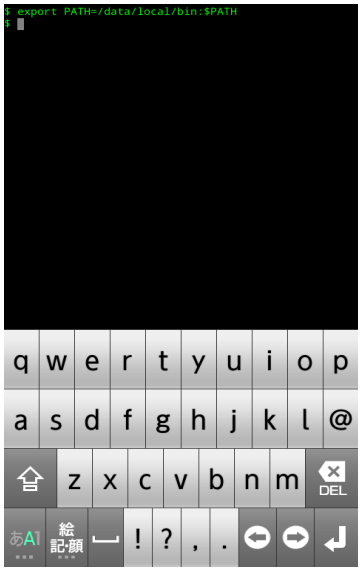
なお、Windows のコマンドプロンプトで Perl を実行したい場合は、以下のインストールができます
・ActivePerl のインストール＜執筆時のバージョン 5.14.2.1402＞ 64bit 使用 (Windows 用)
なお、cygwin をインストールして perl をインストールする場合、cygwin 上で perl 実行ができるので、
Windows プロンプト上で実行必要がなければ特に不要です。
http://www.activestate.com/activeperl
からインストーラをダウンロードしてインストールします。（パスも自動追加されます）

4. Linuxのファイルシステム

今回は、Android OSのベースであるLinux部から、システムの構造を見ていくことを目的としています。

Linuxのシステムを見るためには、Linuxの操作が可能なシェルに入る必要があります。
携帯本体でシェルに入るには、アンドロイドマーケットでフリー配布されているターミナルプログラムを使います。 筆者はJack Palevich氏の Android Terminal Emulator を利用しています。

アプリケーションを起動すると、以下のようにコマンドプロンプトが表示されます



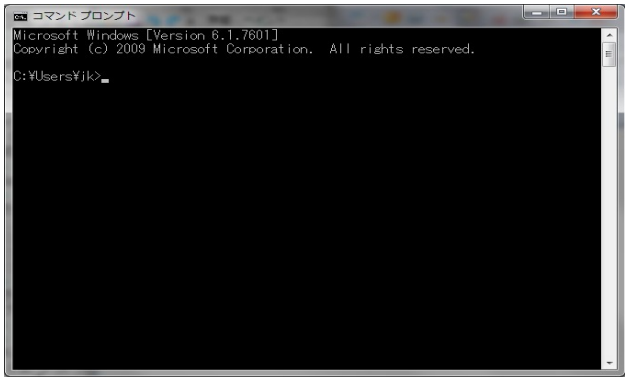
本文上ではこのプロンプトの表示を

```
$
```

と表示します。(緑)

また、先に述べた adb を利用することで、PCと携帯電話をUSBでつないで、PCのコマンドプロンプトから adb shell を実行することで、PCから携帯のシェルに入ることができます
この場合携帯の設定で、アプリケーション－＞開発を開き、USB デバッグをチェックする必要があります。

PCのコマンドプロンプトは、Windows のスタートメニューから、ファイル（cmd.exe）を指定して実行するか、スタートメニューのアクセサリにあるコマンドプロンプトを実行します。



ここでは、PC のコマンドプロンプトの表示を

```
>
```

と表記します。この状態で、adb のシェル を起動します。（青）

```
>adb shell
$
```

>は windows のコマンドプロンプト、$は Linux シェルに入った後のプロンプトを示しています。

＊adb で PC と携帯を接続するためには、PC に adb USB ドライバのインストールが必要です

いづれかの方法により携帯本体のシステムにシェルコマンドでアクセスすることができます。
シェルコマンドは、LINUX（UNIX）シェルコマンドが使えます。

なお、Androidのアプリはアプリごとにユーザーを割り当ててそのユーザーとして起動されています。
上記 Android Terminal Emulatorを起動した場合は、そのアプリのユーザIDでログインした状態に見えます

```
$id
uid=10091(app_91) gid=10091(app_91) group=1015(sdcard_rw),3003(inet)
```

10091 はそのアプリがインストールされたときに確定する任意の番号となっています。

Linuxのファイルシステムは、ファイルの所有者、所有フループ、アクセス権が定められているので、許可されていないファイルへのアクセスはできないようになっています

なお、Cygwin を起動した場合の、プロンプトは、本文では以下のようにあらわします。(橙)

```
$
```

区別の必要のない場合は黒枠となっています。

例）ls -l で表示した例（一部）

```
        ・・・
drwxr-xr-x root    root          2011-07-01 00:00 system
drwxr-xr-x root    root          2011-12-04 00:13 sys
drwxr-x--- root    root          1970-01-01 09:00 sbin
dr-xr-xr-x root    root          1970-01-01 09:00 proc
-rwxr-x--- root    root     1260 1970-01-01 09:00 init.target.rc
-rwxr-x--- root    root    23727 1970-01-01 09:00 init.rc
        ・・・
```

drwxrwxrwx 所有者 所有グループ サイズ 日付 ファイル名（ディレクトリ名）

- その他のユーザーのアクセス権
- 所有グループユーザーのアクセス権
- 所有ユーザーのアクセス権

（上記3つ）r：読み込み w:書き込み x:実行権

- ファイルタイプ（d:ディレクトリ l:リンク 等）“-”が一般のファイル実体

すなわち、ターミナルアプリ＝ユーザーID＝app_91 がアクセスするためには、そのファイルの所有者、 所有グループが app_91 出ない場合は、その他のユーザのアクセス権に rwx が設定されていないと、読み込み、書き込み、実行ができないことになります。

drwxr-x--- root root ..... の設定のディレクトリは、その配下のファイルの存在すら確認することができません。

基本的に、SDカード以外は、読むことはできても書き込み可能なファイル、フォルダは用意されていません。

AndroidSDKで作成するJavaアプリ（apk）自身は、/data/data/(アプリ名)/のフォルダ内にのみ自由にデータのアクセスができる仕組みになっています。（ただしシェルからその内容をみることもできません）

～～～～～～～～～～～～～～～**余談ですが**～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～
/data/data/(アプリ名)領域はそれぞれのアプリのみアクセス可能なため、シェルで見ようとしても見れません。しかし、シェルプログラムである AndroidTerminalEmulator は、自身のアクセス可能なフォルダにはアクセスできるのです。

```
$cd/data/data
$ls -l
opendir failed, Permission denied
```

となるが、ここで

```
$ls -l -d /data/data/jackpal.androterm
drwxr-x—x  app_91   app_91            2011-12-13 01:13  jackpal.androterm
```

となり、/data/data/jackpal.androterm/ フォルダ以下は、このターミナル(app_91)では読み書きが可能であることがわかります。
～～～

一方、adb でシェルに入った場合、

```
>adb shell
$id
id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1009
(mount),1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net_bt),3003(inet)
$
```

のように、ユーザID＝2000(shell)となっています。これは、実はもともとAndroidシステムが adb を接続することを前提に shell ユーザーが用意されています。（システム起動時に adb 接続アプリがshellユーザで起動している）

そして、携帯本体のデバイス領域の 1 つである/data フォルダ内に shell ユーザーが自由にアクセスできる領域を開放しています。それが/data/local/フォルダとなっています

```
$ ls -l -d /data/local
ls -l -d /data/local
drwxrwx--x shell    shell           2011-11-25 02:18 local
$
```

なおこの下には tmp フォルダが同様の権限で作成されていますが、初期状態ではこれらフォルダの中身にファイルは何もない状態となっています。

したがって、PCで adb を経由して本領域にファイルを送ったり、フォルダを作ることも可能となっています。

ところで、先にSDカードはRW可能であると書きましたが、確かにこの shell ユーザもSDカードが書き込めるグループに入っていますが、この shell ユーザーである adb shell からだけ、直接に本体デバイスにファイルを転送することができます。本体デバイスに実行可能なファイルを送り、シェルから実行することも可能となっています。

Sdカードにファイルを転送することも可能ですが、SDカードでは、直接の実行を禁止するようにファイルシステムがマウントされているため実行はできないようになっています。（セキュリティのため）

〜〜〜〜〜〜〜〜〜〜〜〜〜〜シェルからプログラム（コマンド）の実行〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜

実行ファイルを実行するには、フルパス指定するか、ファイルのある位置でカレントパス(./)をつけます。

```
$/data/local/bin/execute
```

（Linux 共通）

あるいは、

```
$cd /data/local/bin/
$./execute
```

なおパス（＄PATH）が設定されている場合は実行ファイル名だけで実行できます

```
$execute
```

パス指定を確認するには、

$echo $PATH で確認できます（あるいは set コマンドですべての環境変数を表示させます）

〜〜〜

5. Linuxコマンドの拡充

以上のように、シェルを起動してLinaxのコマンドを実行することができるのですが、ファイル操作を行うLinuxの基本コマンド等が、Android端末では十分に用意されているわけではありません。

```
$ echo $PATH
echo $PATH
/sbin:/vendor/bin:/system/sbin:/system/bin:/system/xbin
```

となっていてサポートしている基本コマンドは、/system/bin に入っています。

```
$ls -l /system/bin
```

とするとその内容がリスト表示されますが、そのうち基本的なファイル操作等を行うコマンドは、

```
lrwxr-xr-x root     shell              2011-07-01 00:00   ○○○○ -> toolbox
```

と書かれているコマンドであることがわかります。
これは、その実体が、toolbox という実行ファイルであるリンクファイルとして実行ファイルを用意していることを示しています。

しかし、ファイルをコピーしたりする cp コマンドも見つかりません。

そこで、コマンドを拡張するために、busybox というツールを用意します。busybox は、http://busybox.net から入手することができます。
IS05 の CPU（MSM 8655）情報を調べると

```
$ cat /proc/cpuinfo
cat /proc/cpuinfo
Processor       : ARMv7 Processor rev 1 (v7l)
BogoMIPS        : 163.57
Features        : swp half thumb fastmult vfp edsp neon vfpv3
CPU implementer : 0x51
CPU architecture: 7
CPU variant     : 0x1
CPU part        : 0x00f
CPU revision    : 1
Hardware        : SHARP DECKARD AS32
Revision        : 0000
Serial          : 0000000000000000
$
```

命令セットは　ARMv71　であることが分かりますが、上記サイトで実行バイナリ提供は V6l までであったので、提供ソースからコンパイルして実行ファイルを生成する必要があるようです。
そのためには、Linux 環境か、Windows なら AndroidNDK+CygWin 環境が必要となります。
実際 busybox は専用の make ファイルで構築するため、AndroidNDK+CygWin 環境での環境だとうまく make できませんでした。

ただ、公開されている、ARM6l 用の実行ファイルでも下位互換性があるのか、実際には動作しているのでそれを用いてもよいでしょう。あるいは、一般公開されている Android アプリ（apk）に含まれている場合もあるので、それを取得して、ZIP 展開してみるとその中にある実行ファイルを取り出すことも可能です。
Root 取得関連アプリに含まれている場合が多いです。

さて、busybox 実行バイナリが用意できたら、それを本体に転送して使えるようにします。

携帯にデータを転送して実行するには、/data/内にユーザーに開放された領域に実行ファイルを転送する必要があります。一般に/data/local ディレクトリ以下がユーザーに公開されています。ただし、ここは adb で接続した場合のユーザー（shell）しかアクセスできません。

そこで、adb を使って転送します。（コマンドプロンプトで busybox のあるフォルダに移動しておきます）
携帯アプリの Android Terminal Emulator では初期コマンドに/data/local/bin にパスを通すように設定されていることから、/data/local/bin にコマンドを追加していきます。（フォルダがないので作成します）

```
>adb shell mkdir /data/local/bin
>adb push busybox /data/local/bin
>adb shell chmod 755 busybox
```

では実際に動かしてみます。adb shell ではパスに上記格納フォルダが追加されていないので追加します

```
>adb shell
$export PATH=/data/local/bin:$PATH
$
```

これで busybox が起動できます。

```
$ busybox
busybox
BusyBox v1.18.4 (2011-08-28 00:30:13 JST) multi-call binary.
Copyright (C) 1998-2009 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: busybox --list[-full]
   or: function [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
     [, [[, adjtimex, awk, base64, basename, bbconfig, blkid, blockdev,
     bunzip2, bzcat, bzip2, cal, catv, chgrp, chroot, cksum, clear, cmp,
     comm, cp, cpio, cut, date, dd, df, diff, dirname, dmesg, dos2unix, du,
     dumpkmap, echo, ed, egrep, env, expand, expr, false, fdisk, fgrep,
     find, findfs, flash_eraseall, flash_lock, flash_unlock, flashcp, flock,
     fold, free, freeramdisk, fsck, fuser, getopt, grep, gunzip, gzip, hd,
     hdparm, head, hexdump, hostid, id, install, iostat, kill, killall,
     killall5, length, less, ln, loadkmap, losetup, ls, lsattr, makedevs,
     md5sum, mkdir, mkdosfs, mke2fs, mkfifo, mkfs.ext2, mkfs.vfat, mknod,
     mkswap, mktemp, more, mount, mountpoint, mpstat, mt, mv, nanddump,
     nandwrite, netstat, nice, nohup, od, patch, pgrep, pidof, pkill,
     printenv, printf, ps, pwd, rdate, readlink, realpath, renice, reset,
     resize, rev, rfkill, rm, rmdir, script, scriptreplay, sed, seq,
     setkeycodes, setsid, sha1sum, sha256sum, sha512sum, sleep, sort, split,
     start-stop-daemon, stat, strings, sum, swapoff, swapon, switch_root,
     sysctl, tac, tail, tar, tee, test, time, timeout, top, touch, tr, true,
     tty, ttysize, ubiattach, ubidetach, umount, uname, unexpand, uniq,
     unix2dos, unzip, uptime, usleep, uudecode, uuencode, vi, watch, wc,
     which, who, whoami, xargs, yes, zcat
$
```

といったヘルプがでてきます。

$busybox [コマンド名] パラメーター　で上記コマンドが使用できるようになりますが、いちいち busybox のコマンドを入れるのも面倒なので、各コマンドを展開します。

展開方法にはヘルプにあるように、各コマンドを完全に展開インストールもできるが、toolbox と同じように、リンクをはることでコマンドの実体を busybox のみとすることができるのでこの方法で展開します。

展開にはシェルのスクリプトを利用します。以下のようなシェルスクリプトファイルを用意して、busybox と同じフォルダにいれて実行権を与えます。

binst ファイル（スクリプト）

```
#!/system/bin/sh
for cmd in `/data/local/bin/busybox --list`
do
  ln -s /data/local/bin/busybox /data/local/bin/${cmd}
done
```

```
>adb push binst /data/local/bin
>adb shell chmod 755  binst
```

これで各追加コマンドをそれぞれ実行できるようになります。

ここで、追加したコマンドで、ls コマンドはカラー対応のシェル用にカラーコードが追加されています。
携帯にインストールした Android Terminal Emulator でみるとデータの種類ごとに色分けされています。
しかし、adb shell で起動した場合はコントロールコードがそのままでてきて、見づらくなってしまいます。
そこで ls コマンドを ls2 とかに変更しておき、adb shell からは今までの ls を使えるようにします

```
$mv /data/local/bin/ls /data/local/bin/ls2
```

（あるいは rm コマンドで削除します）

☆スクリプトファイルの注意点
Linux のファイルの改行はLF（0x0A コード '\n'）となっています。Windows はCR(0x0D '\r') + LFです。
したがって、Windows でスクリプトファイルを作成するときは、CR が入らないように修正が必要となります。

～～～～～～～～～～～～～～～～ カスタムフォントの導入 ～～～～～～～～～～～～～～～～～～～～～～～～

携帯電話の本体領域で開放されている領域は/data/local であると述べましたが、Sharp 製のスマートフォンには、もうひとつユーザーに開放されている領域があります。それは、/data/fonts フォルダです。

```
$ ls -d -l /data/fonts
ls -d -l /data/fonts
drwxrwxrwx root     root              2011-11-21 10:22 fonts
```

ここには、ユーザーの任意のフォントを追加することで、システムフォントに追加できる仕組みが入っています。任意のフォントファイル（xxxxxxxx.tff）を、customfont.tff という名前にかえて、このフォルダに入れるだけです。（ttf以外にttc、otfも可能とのこと）1つだけフォントを追加することができるというものです。
（入れ替えればいくらでもできるということだが、Root 取得しなくても追加できるところがよい）

さてこのフォルダ、どのユーザでも自由に読み書き、実行までできるフォルダになっています。
ということは adb でなくても、携帯本体のみでアクセスしたときに、実行ファイルの格納実行ができるということです。（まあ、一度/data/local/bin のように adb でその他ユーザも書き込めるフォルダを作ってしまえば、良いのですが。。）
adb を一度も接続していなくて、adb 環境もない場合、携帯本体のみで調べる場合、このフォルダを利用するというのも、ひとつの手ではあります。
～～～

6．データ格納デバイス

携帯のデータ格納デバイスには、フラッシュデバイス、RAM デバイスそして SD カードデバイスがあります

携帯本体のシステムは、フラッシュデバイスに格納されています。確認するには、以下のようにします

```
$ cat /proc/mtd
cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00b00000 00020000 "boot"
mtd1: 01e00000 00020000 "cache"
mtd2: 00100000 00020000 "misc"
mtd3: 00b00000 00020000 "recovery"
mtd4: 00f00000 00020000 "ipl"
mtd5: 19a00000 00020000 "system"
mtd6: 00300000 00020000 "persist"
mtd7: 00300000 00020000 "log"
mtd8: 00300000 00020000 "battlog"
mtd9: 00200000 00020000 "calllog"
mtd10: 01200000 00020000 "ldb"
mtd11: 1a5a0000 00020000 "userdata"
$
```

本体のフラッシュデバイスは12個の区画(mtd0~11)に区切られ、それぞれに名前が定義されています

”ipl”は initial program loder つまりパソコンでいう Bios に相当するプログラムが入っていると考えられます。機種によってはない場合もあります。 ”boot”あるいは”recovery”のシステムを起動するための処理が入っている。

”boot”と”recovery”は、ほぼ同じ動作を行うための領域で、Linux システムを立ち上げるためのデータで、その中は、ブートヘッダとカーネルイメージ領域、RamDisk 領域、そして SecondStage という拡張領域に分けられて格納されています。（実際はこれらの結合イメージを UBIFS 形式にパックされた形になっています）
なお SecondStage 領域は将来のオプションで現在は何もないっていない場合がほとんどのようです。
ブートヘッダは領域を管理するヘッダ、カーネルイメージは Linux 立ち上げの実行コード、RamDisk 領域は Linux の初期立ち上げ時に用意する初期 RamDisk（initramfs）にルート展開させるファイルシステムのデータが入っています。なお RAM 領域は cpio 形式に圧縮されています。

カーネルイメージは、ブートコード、セットアップルーチン（展開ルーチン）、圧縮カーネルの構造になっています。

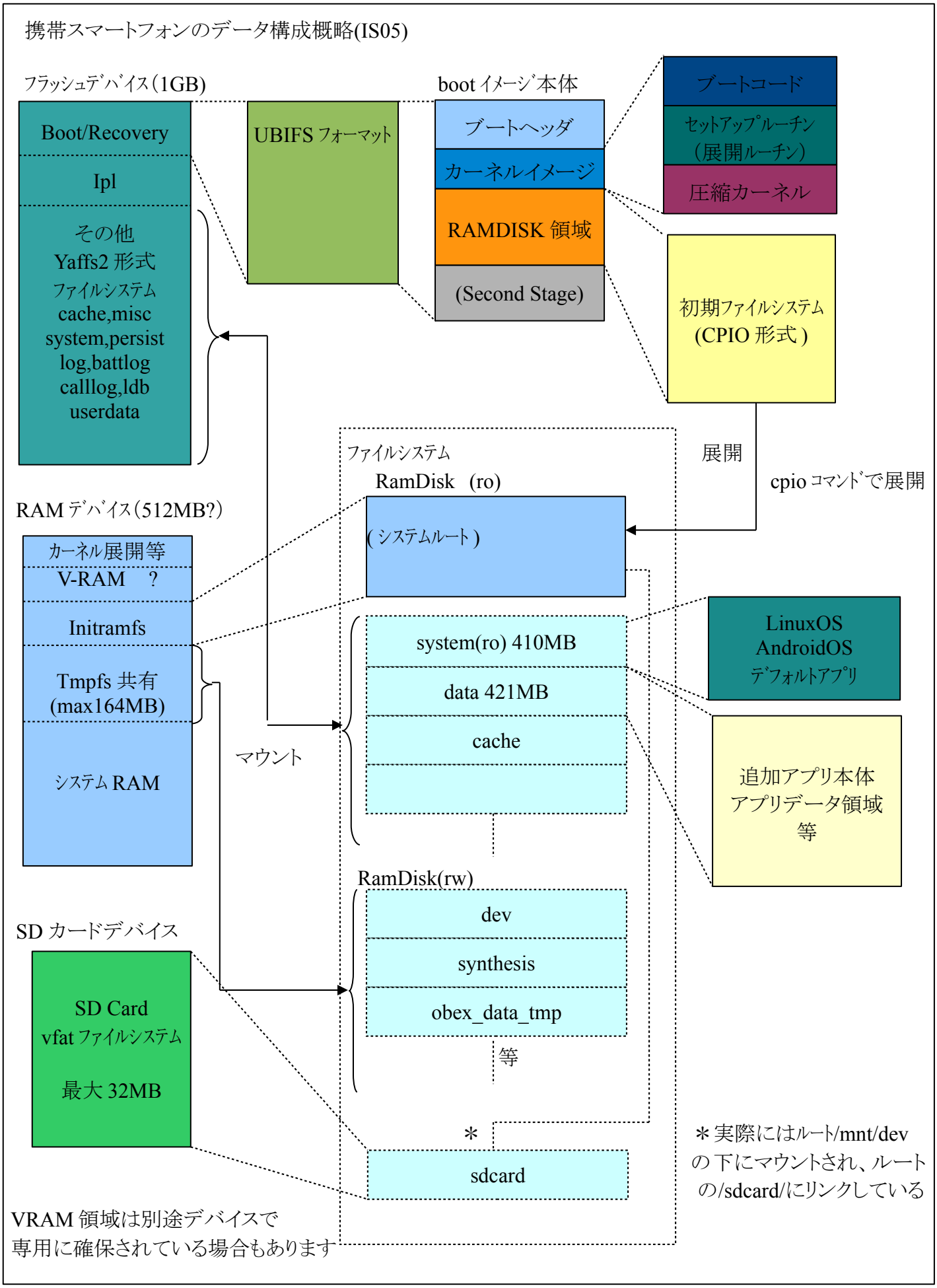
通常パソコン等の OS として起動する Linux の場合は、この初期立ち上げ後、初期 RamDisk に展開されたルートを本来の HDD デバイスにマウントしなおすようになっているが、携帯スマートフォンの場合はこのルート展開したまま運用します。この領域は展開後 ro(ﾘｰﾄﾞｵﾝﾘｰ)に再マウントされます。

”boot”と”recovery”の違いは、本機では、オールリセットのときのシステム再構築のときに、本領域のシステムを利用してシステムの再構築をしているようである。 通常は ”boot”領域から起動される。

そして、その他の領域は、上記 RamDisk 領域から展開されたルートシステムをベースに、それぞれのフォルダとしてマウントされていきます。Yaffs2 でディレクトリにマウントして直接 RW 可能な形式となっています

なお、システム上必要な RamDisk として tmpfs が 164MB 用意され、複数のフォルダのファイルシステムで共有して利用されます。

また、SD カードもファイルシステムのひとつとして、マウントされます


携帯スマートフォンのデータ構成概略(IS05)
フラッシュデバイス(1GB)
Boot/Recovery
Ipl
その他
Yaffs2 形式
ファイルシステム
cache,misc
system,persist
log,battlog
calllog,ldb
userdata
UBIFS フォーマット
boot イメージ本体
ブートヘッダ
カーネルイメージ
RAMDISK 領域
(Second Stage)
ブートコード
セットアップルーチン
(展開ルーチン)
圧縮カーネル
初期ファイルシステム
(CPIO 形式)
展開
cpio コマンドで展開
ファイルシステム
RamDisk (ro)
(システムルート)
RAM デバイス(512MB?)
カーネル展開等
V-RAM ?
Initramfs
Tmpfs 共有
(max164MB)
システム RAM
マウント
system(ro) 410MB
data 421MB
cache
LinuxOS
AndroidOS
デフォルトアプリ
追加アプリ本体
アプリデータ領域
等
RamDisk(rw)
dev
synthesis
obex_data_tmp
等
SD カードデバイス
SD Card
vfat ファイルシステム
最大 32MB
＊
sdcard
＊実際にはルート/mnt/dev
の下にマウントされ、ルート
の/sdcard/にリンクしている
VRAM 領域は別途デバイスで
専用に確保されている場合もあります

7. Boot/Recovery ブートイメージの構造（分解と再構築）

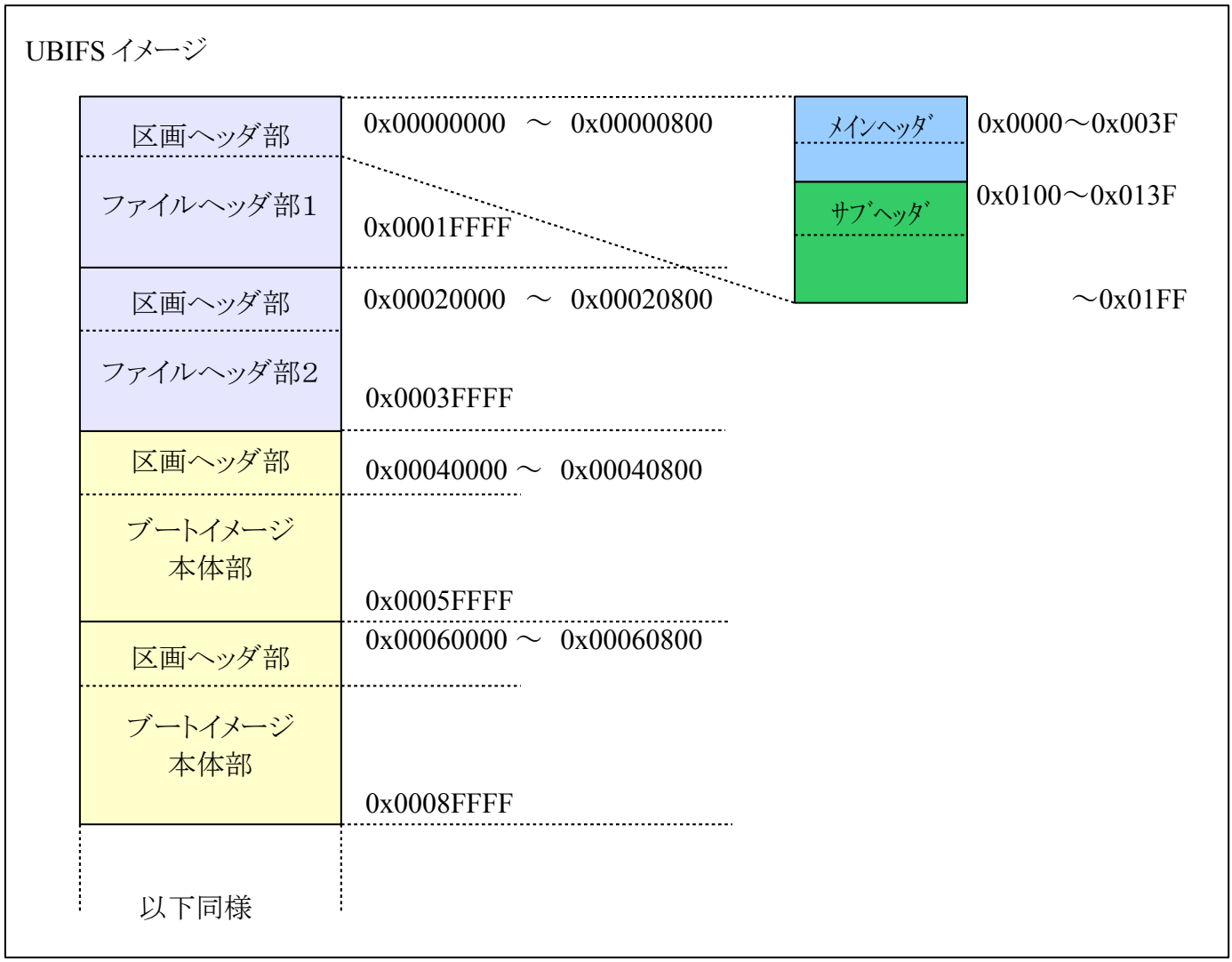
ここで、Boot/Recovery ブートイメージの構造についてその詳細を見てみます。

実際に携帯本体のフラッシュデバイスからデータを取り出してみます。
そのためには、実は Root 権限が必要となります。Root 権限の取得についてはここでは触れません。
フラッシュデバイスの分割されたブロックからデータを取り出すためには、Root 権限で携帯本体の
シェルに入ります（adb shell または本体の Terminal）

```
# cat /dev/mtd/mtd0 > /sdcard/boot.img
```

(#は Root 時のプロンプト)

この取得したデータは、以下のような UBIFS フォーマットになっています。



このファイルシステムは一定サイズ（デバイスの書き込み単位 0x20000）で区切られたブロックそれぞれ区画のヘッダ情報を持ちます。
区画情報ヘッダは先頭から 0x00000000 ～ 0x00000030 にのメインヘッダと 0x00000100 ～ 0x00000130 にサブヘッダを持つ構造になっています。先頭から2区画はファイルヘッダ部となっているようです

先頭のヘッダ部1，2の 256KByte の後（0x00040000～）、128Kbyte(0x00020000)単位の区画で、区切られた領域に区画ヘッダ 2KByte(0x800)とブートイメージ本体が格納されています。
UBIFS の詳細な構造はおいておき、単なるサイズ分割でブートイメージ本体を取り出すことができます。

このUBIFSフォーマットは、フラッシュデバイスの書き込みに適したブロックで分割されており、
さまざまなファイルシステム(Ext3,VFATなど)をフラッシュデバイスの書き込みに最適化して格納する
ため、ファイルシステムとデバイスとの間に入って変換を行うように考えられたものです。

携帯の本体のシステム(/system)やアプリデータ部（/data）はYaffs2というファイルシステムで自体が、フラッシュデバイスに最適な仕組みとして、アクセスを高速にできるようになっています。フラッシュデバイスに直接アクセスできるようになっています。SDカードはPCとアクセスしてメモリカード相当に見せるためVFAT形式となっています。

そこでブートイメージ本体を取り出すために、**unubinize.pl**というプログラムがインターネットを検索すると入手することができるので、これを利用します。（これもRoot関連のページで再配布されていると思います）

ただし、これはperlプログラムなのでPerlを実行できる環境をインストールしておきます。

そのために、CygwinにPerlをインストールします。やり方は簡単。Cygwinのsetup.exeを起動して
追加のモジュールを選択します。

Devel >subversion-perl:A version control system(perl bindings)

の項目を選択して実行します。

Perlをインストールすると、Cygwinのシェルプロンプトで

```
$perl -v
```

と実行することで、Perlのバージョン情報が表示されるようになります。

さて、unubinize.plを入手したら、携帯から取得したデータ(boot.img)とともに,、任意のフォルダに入れます。
例：/home/(user)/workを作成しておきます。

先ほどブートデータは携帯のSDカードに取得したのでPCに持ってきておきます。
(Windowコマンドプロンプトから　adbを起動します）

```
>cd  C:￥（Cygwinフォルダ）￥home￥(user)￥work
>adb pull /sdcard/boot.img
```

そして、再びCygwinのシェルで

```
$cd /home/(user)/work
$./unubinize.pl boot.img
```

を実行すると、boot.img.outというファイルが生成されます

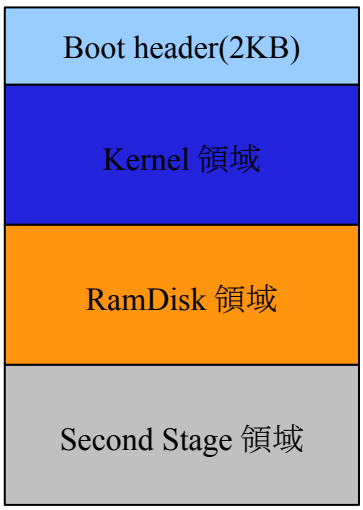
unubinize.pl は以下のようにファイルをサイズ分割＆結合を行っている処理となっています

<unubinize.pl >

```
#!/usr/bin/perl

$fname = shift;
die unless -f "$fname";

open(IN, "< $fname") || die;
binmode(IN);

open(OUT, ">$fname.out") || die;
binmode(OUT);

die unless seek(IN, 0x040000, 0);

while (!eof(IN))
{
        die unless read(IN, $data, 0x000800) == 0x000800;

        $n = read(IN, $data, 0x01f800);
        die unless $n > 0;
        die unless syswrite(OUT, $data) == $n;

        last unless $n == 0x01f800;
}

close(IN);
close(OUT);

exit 0;
```

簡単なファイルなのでテキストエディィタで作成することもできると思います

さて、生成された、ブートイメージ本体boot.img.outはどのような構造になっているのかというと、



1 page　page 単位はフラッシュデバイスの書き込み単位

n pages　n = (kernel_size + page_size - 1) / page_size

m pages　m = (ramdisk_size + page_size - 1) / page_size

o pages　o = (second_size + page_size - 1) / page_size

*Kerner 領域と RamDisk 領域は必須（サイズ≠0）
**Second Stage 領域は存在しない場合はサイズ＝0となる

以上のようなに、Page 単位で分割された領域に Kernel 情報と RamDisk 情報が格納されています。Second Stage 領域は将来の拡張用で現在は使われていないようです。

Boot header 情報は、以下のような構造で各ブロックの情報などが格納されています

```
<Boot header 構造体>
struct boot_img_hdr
{
  unsigned char magic[BOOT_MAGIC_SIZE];                    BOOT_MAGIC_SIZE は 8

  unsigned kernel_size;  /* size in bytes */
  unsigned kernel_addr;  /* physical load addr */

  unsigned ramdisk_size; /* size in bytes */
  unsigned ramdisk_addr; /* physical load addr */

  unsigned second_size;  /* size in bytes */
  unsigned second_addr;  /* physical load addr */

  unsigned tags_addr;    /* physical addr for kernel tags */
  unsigned page_size;    /* flash page size we assume */
  unsigned unused[2];    /* future expansion: should be 0 */

  unsigned char name[BOOT_NAME_SIZE]; /* asciiz product name */    BOOT_NAME_SIZE は 16

  unsigned char cmdline[BOOT_ARGS_SIZE];                   BOOT_ARGS_SIZE は 512

  unsigned id[8]; /* timestamp / checksum / sha1 / etc */
};
```

先頭の BOOT MAGIG には、'ANDROID!' の文字列が格納されることになっています。

そしてこの、ブートイメージ本体から、Kernel 本体と、RamDisk 本体を取り出すには、**split_bootimg.pl** というプログラムを使います。これもインターネットを検索すると見つかると思います。

先ほど取り出したブートイメージ本体データ（boot.img.out）とともに、split_bootimg.pl を格納したフォルダで Cygwin のプロンプトから実行させます。

```
$./split_bootimg.pl boot.img.out
```

すると

```
$./split_bootimg.pl boot010101.bin.out
Page size:      2048 (0x00000800)
Kernel addr:    2129920 (0x00208000)
Kernel size:    8434976 (0x0080b520)
Ramdisk addr:   18874368 (0x01200000)
Ramdisk size:   312832 (0x0004c600)
Board name:
Command line: console=ttyDCC0 androidboot.hardware=qcom
```

と表示されます。
このとき表示される情報は、再構築して書き戻す際に必要となる情報です
なお、このときの Kernel addr: 0x00208000 から 0x8000 を引いた値が、ベースアドレスとなります。

そして、
boot.img.out-kernel　　（カーネルイメージ本体）
boot.img.out-ramdisk.cpio（RamDisk 情報本体）

それぞれサイズは split_bootimg.pl 実行時に表示された値になっています（ページ単位ではない）

カーネルイメージ本体は、Linux の初期展開を行うネイティブなプログラムで、Linux システムのブート処理を行い、メモリ（RAM）上に初期 RamDisk（initramfs）を作成して、RamDisk 情報本体に格納されているファイルシステムを展開します。

この RamDisk 情報本体は、cpio 形式でディレクトリ構造を丸ごとイメージ化されています。
この中をみることで、実質上の携帯のファイルシステムの Root 部がわかります。
そこでこの、RamDisk 情報本体（boot.img.out-ramdisk.cpio）を展開するには、Linux の **cpio** コマンドを用います。これは busybox にも入っていますが、Cygwin でも入っています。

RamDisk 情報本体（boot.img.out-ramdisk.cpio）のあるディレクトリにフォルダを作ってその中に展開します

```
$mkdir ramdisk
$cd ramdisk
$cat ../boot.img.out-ramdisk.cpio | cpio -i
```

これで、/ramdisk フォルダに展開されます。その内容を見てみると、

```
$ ls -l
-rw-r--r-- shell    shell        8499 2011-12-08 15:04 ueventd.rc
-rw-r--r-- shell    shell           0 2011-12-08 15:04 ueventd.goldfish.rc
drwxr-xr-x shell    shell             2011-12-08 15:04 system
drwxr-xr-x shell    shell             2011-12-08 15:04 sys
drwxr-x--- shell    shell             2011-12-08 15:04 sbin
drwxr-xr-x shell    shell             2011-12-08 15:04 proc
-rwxr-x--- shell    shell       47548 2011-12-08 15:04 initlogo.rle
-rwxr-x--- shell    shell        1260 2011-12-08 15:04 init.target.rc
-rwxr-x--- shell    shell       23727 2011-12-08 15:04 init.rc
-rwxr-x--- shell    shell        7123 2011-12-08 15:04 init.qcom.sh
-rwxr-x--- shell    shell        9097 2011-12-08 15:04 init.qcom.rc
-rwxr-x--- shell    shell        1678 2011-12-08 15:04 init.goldfish.rc
-rwxr-x--- shell    shell      101672 2011-12-08 15:04 init
drwxr-xr-x shell    shell             2011-12-08 15:04 dev
-rw-r--r-- shell    shell         118 2011-12-08 15:04 default.prop
drwxrwx--x shell    shell             2011-12-08 15:04 data
$
```

これが、携帯本体のファイルシステムのルート（rootfs）として RamDisk である initramfs に展開されています。
その中に作られているディレクトリは、sbin に初期ファイルがいくつか格納されている以外は空のフォルダとなっています。
そして展開された初期ファイルシステムの情報にしたがってシステムを構築していきます。
このシステムの骨格（スケルトン）にフラッシュデバイスのさまざまな区画にわけられたファイルシステム形式（Yaffs2）のデータがフォルダに結合（マウント）されていきます。

なお、システムの構築が完了するとルートにある RamDisk は読み取り専用にリマウントされるので、通常この領域を書き換えることはできません。

さて、この分解された、RamDisk とカーネルイメージを再び再構築する方法を見ていきます。

再構築して、フラッシュデバイスに書き込むことで、初期フォルダ構成を変更することで初期立ち上げ処理を変更したりすることができます。

展開された Ramdisk のフォルダと、もとの Kernel イメージ本体のデータから元に戻していきます。

まず、RamDisk のフォルダを cpio 形式に変換します。これには Android ソースに添付の **mkbootfs** コマンドを利用しますが、携帯本体にも Linux にも Cygwin にもありません。インターネットを検索すると、大抵は PC の Linux 上で動く実行ファイルしか出回っていません。

そこで、Android のソースを取得して、ソースから Cygwin 上で gcc コンパイルして Cygwin で動く実行ファイルを作りたいと思います。

Android のソースの取得は、Android Open Project サイトにその方法が公開されています。
http://source.android.com/source/downloading.html

サイトの情報に従えばいいのですが、CygWin で取得するには、いくつかのツールが足りないので、インストールする必要があります。Cygwin の setuup.exe を実行して、
以下の追加モジュールを選択してインストールします。

Devel >mercurial:Python based distribute version controle system(DVCS)
Devel >git:Fast Vaersion Contlole System -core files
Devel >git-completion:Fast Vaersion Contlole System -git bash completion
Devel >git-gui:Vaersion Contlole System -git-gui viwer
Devel >subversion-perl:A version control system(perl bindings) (Perl インストール済みなら不要)
Archive > zip: Info-ZIP compression utility
Devel > bison: A parser generator that is compatible with YACC
Devel > flex: A fast lexical analyzer generator
Devel > gperf: Perfect fash function generator
Devel > ruby: Interpreted object-oriented scripting language
Python > python: Python language interpeter
Util > gnupg: GNU's tool for secure communication and data storage
Web > curl: Multi-protocol file transfer command-line tool

ここでさらに、以下の処理を実行しておきます。
a. 全ての Cygwin を停止。
b. Windows のコマンド・プロンプトを起動し、以下を実行。

```
> cd <Cygwin 導入フォルダ>\bin
> ash.exe
$ /usr/bin/rebaseall
```

を実行します。(これはエラーが発生したための回避方法です)

そして、cygwin を再起動し、
準備ができたら、Cygwin を立ち上げ、サイトの説明のとおり、 取得のためのコマンドを、取得します。

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
```

と取得コマンドフォルダの作成とパスを設定しておきます
そして

```
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

と取得したファイルに実行権を与え、
ソース取得フォルダを作り、その中に移動しておきます。

```
$ mkdir SRC_ANDROID
$ cd SRC_ANDROID
```

そして、最新のバージョンを取得するマニフェスト情報を構築するため

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

を実行します（ここで名前と Email 入力を聞かれるので入力します）

特定のバージョンを取得する場合 -b を用いて

```
$ repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1
```

のように実行します。

その後、

```
$repo_cync
```

でソースの取得が開始します。数時間ほどかかる場合があります。android-4.0.1_r1 で9 GB ほどあります。

さて、ソースの取得ができたら、mkbootfs を探してみます。
ソースフォルダの/system/core/cpio というフォルダに mkbootfs.c ファイルが見つかります。
ソースをみると、
#include <private/android_filesystem_config.h>
と書いているので、 ソースフォルダの/system/core/include/private にある android_filesystem_config.h も取得します。 mkbootfs.cを任意のフォルダ（例：/home/(user)/mkbootfs/)に、インクルードファイルはその下の/private/フォルダにいれておきます。

と一緒に、Cygwin の任意のフォルダに持ってきておきます。

そしてその任意のフォルダ（例：/home/(user)/mkbootfs/）にて、

```
$gcc -I./ mkbootfs.c
```

とコンパイラを実行すると同一フォルダに a.exe というファイルができます。これを mkbootfs.exe とします。
これで Cygwin で実行可能な mkbootfs 実行ファイルができました。

そして、元に戻って、この実行ファイルを、先の展開された ramdisk フォルダとカーネルイメージ本体のデータのあるフォルダにコピーして、そのフォルダに移動し、ramdisk フォルダをイメージ化します

```
$cd /home/(user)/work
$./mkbootfs ramdisk > ramdisk.img
```

そして、このできた Ramdisk イメージとカーネルイメージ本体を結合してブートイメージ本体に戻すには、
mkbootimg というツールをつかいます。
これも mkbootfs と同じく Cygwin で実行するには、Android ソースから
/system/core/mkbootimg/mkbootimg.c と bootimg.h
/system/core/libmincrypt/sha.c
/system/core/include/mincrypt/sha.h
を取得して任意のフォルダに入れます（例：/home/(user)/mkbootimg/）
sha.h は/mincrypt/フォルダの下に入れます。（例：/home/(user)/mkbootimg/mincrypt） そして、

```
$gcc mkbootimg.c sha.c
```

で実行ファイル a.exe ができるので、mkbootimg.exe として Ramdisk イメージとカーネルイメージ本体のフォルダにコピーして、そのディレクトリに移動して、実行します。

```
$cd /home/(user)/work
$./mkbootimg --kernel  boot.img.out-kernel
        --ramdisk ramdisk.img
        --cmdline "console=ttyDCC0 androidboot.hardware=qcom"
        --base 0x00200000
```

複数行に記述しているが実際のコマンド上はスペースでつながった1行で連続しています。

実行すると、結合されたブートイメージ本体ができます。Kernel はとくに変更の必要もないのでもとのファイルを使っています。このときの引数に、split_bootimg.pl 実行時に表示された情報と、ベースアドレスを使います。
実行すると tmpboot.img というイメージファイルができます。これを UBIFS 形式に格納したら、もとのフラッシュデバイスから引き出した形に戻ります。

この UBIFS に変換するには、**ubinize** コマンドを使います。Linux では追加することができるコマンドだそうですが、Cygwin の環境で実行するコマンドが見つかりません。

そこで、http://www.linux-mtd.infradead.org/faq/ubifs.html の How can I create UBIFS images?という項目を参照したところ、http://git.infradead.org/mtd-utils.git にソースが公開されています。

/mtd-utils.git/ubi-utils/ のフォルダから
ubinize.c、dictionary.c、libubigen.c、ubiutils-common.c、libiniparser.c を取得して(任意のフォルダ)に、
/mtd-utils.git/ubi-utils/include のフォルダから
dictionary.h 、libiniparser.h、libubi.h、libubigen.h、ubiutils-common.h を取得して(任意フォルダ)/include/の下に、
/mtd-utils.git/lib のフォルダから
libcrc32.c を取得して(任意のフォルダ)に、
/mtd-utils.git/include のフォルダから
common.h、crc32.h、mtd_swab.h、xalloc.h を取得して(任意フォルダ)/include/の下に、
/mtd-utils.git/include/mtd のフォルダから
mtd-abi.h、mtd-user.h、ubi-media.h、ubi-user.h を取得して(任意フォルダ)/include/mtd/の下に、

それぞれ取得したファイルを格納して、
任意フォルダ/include/にヴァージョン情報を定義(適当)した version.h

```
<version.h>
#define VERSION "TestVersion"
```

と
(任意フォルダ)/include/asm/に byteorder.h を以下の内容で作成して格納した。

```
<byteorder.h>
#include <sys/types.h>

typedef __signed__ char __s8;
typedef unsigned char __u8;

typedef __signed__ short __s16;
typedef unsigned short __u16;
typedef unsigned short __be16;

typedef __signed__ int __s32;
typedef unsigned int __u32;
typedef unsigned int __be32;

typedef __signed__ long long __s64;
typedef unsigned long long __u64;
typedef unsigned long long __be64;
```

これは正式の byteorder.h をインクルードすると派生しすぎたので、必要な部のみ記載したものを作成した。

コンパイルは、(任意のフォルダ)の下で、

```
$gcc -I./include dictionary.c libcrc32.c libiniparser.c libubigen.c ubinize.c ubiutils-common.c
```

を実行すると、a.exe ができるので、これを ubinize.exe とします。
これで Cygwin で動作する ubinize コマンドができたので、実行します。

実行のためには、以下のコンフィグファイルを作り、実行ファイルとともに、作業フォルダにコピーします

```
＜ubi.cfg の内容＞
[boot]
mode=ubi
image=tmpboot.img
vol_id=0
vol_size=200MiB
vol_type=dynamic
vol_name=boot
vol_flags=autoresize
```

そして、tmpboot.img のあるフォルダで、

```
$./ubinize -o new_boot.img -p 128KiB -m 2048 -O 256 ./ubi.cfg
```

と実行すると、new_boot.img というフラッシュデバイスに書き込み可能なデータができます。

これを、実際に携帯本体に書き戻すには、携帯本体にこの書き込みデータを転送して、**flash_image** というコマンドで書き込みをおこないます。実行ファイルがインターネットで検索できると思います。
（Android 1．5まで標準添付の実行ファイルだったようです）

なお、ソースファイルは、リカバリキットを公開しているソースに見つかりました。
https://github.com/packetlss/android_bootable_recovery/tree/eclair/mtdutils から
flash_image.c mtdutils.h mtdutils.c （オリジナルは Android OpenSource）
Android ソース（4.0）の Android-SRC/system/core/include/cutils/ から
log.h logd.h uio.h logger.h
Android ソース（4.0）の Android-SRC/system/core/liblog/ から
logd_write.c
を取得して、AndroidNDK でコンパイルすることで arm 用の実行ファイルはできました。（C ファイル3つ）
（ソース修正は不要、cutils のインクルードファイルは/jni/cutils/フォルダに入れ、その他は/jni/の下）

```
$cd (任意のフォルダ：jni のフォルダのある場所)
$ndk-build
```

取得した実行ファイルを携帯本体に転送して、実行権を与えます。そして、書き込みデータを指定して実行しますが、このコマンドの実行には。Root 権限が必要となります。
例：/data/local/tmp/などに転送します。（SD カードではコマンドは実行できません）

```
>adb push flash_image /data/local/tmp
>adb shell chmod 755 /data/local/tmp/flash_image
```

さらにフラッシュデバイスは、デバイスの書き込みロック（NAND Lock）が施されているため、それを解除しないと書き込めないようになっています。
Root の取得と、フラッシュデバイスの NandLock 解除は、後ほどの話として、とにかく解除を実行してから、
書き換え実行は、/data/local/tmp に移動してから、

```
#./flash_image recovery new_boot.img  (Recovery 領域に書き込むとき)
#./flash_image boot new_boot.img （Boot 領域に書き込むとき）
```

で実行されます。

flash_image [デバイス領域名] 書き込みイメージファイル
デバイス領域名は cat /proc/mtd で表示される区画ごとの名前となっています。
最初に取り出すときと、最後に書き戻すときには Root 権限が必要ということになります。

なお、Recovery 領域に書き込んだ場合 Recovery 領域から起動するには、

```
#reboot recovery
```

とします。

＊注意　フラッシュデバイス領域の書き換えを行うと、メーカー、携帯キャリヤのサポートは受けられなくなります。また、書き換え失敗した場合の保証もありませんので、自己責任で行ってください。

Recovery 領域について

さてここまでで、boot 領域のブートイメージについて述べましたが、recovery 領域のブートイメージも同じ形式で格納されていてほとんど同じ情報が格納されています。

実際展開してみると、Recovery 領域の初期処理 init.rc は通常の処理より簡略されており、recovery プログラムが実行されるようになっています。
これは、携帯の設定メニューで、システムを初期化するオールリセットを実行したときに利用されるものと思われます。

シャープ製のスマートフォンでは、直接リカバリ領域がから起動するモードはないということらしいです。
他社の場合、電源とホームを押しながら立ち上げると、ブート起動の選択を行うメニューになるものもあるようです。

なお、adb 接続したときの機能で、
>adb reboot [bootloader|recovery] というコマンドがサポートされていますが、実際に、

```
>adb reboot  recovery
```

を実行すると、

警告マーク（三角！）とドロイドマークが表示されてその先に進まなくなります。
これは、安易にリカバリ領域で起動して、データがクリアされてしまわないように防御していると思われます。

>adb reboot だと通常に再起動します

リカバリ領域に変更ブートを書き込んだ場合は、ルート権限で

```
#reboot recovery
```

を実行することで、リカバリ領域のブートで起動することができます。
(もしかしたらオールクリアを実行するとリカバリ領域の起動になるのか、何らかの判断が入っているのかもしれません)
リカバリ領域にはオールリセット時のシステム初期起動時の画像が入っていました

＊なお作成された UBIFS フォーマットは、区画ﾍｯﾀﾞのﾒｲﾝﾍｯﾀﾞに構築ランダムな数値を格納する領域(image_seq)が存在するため、同じデータで作り直しても同じ内容にはなりません。

＊＊cygwin の実行用ファイルは、Windows のプロンプトで動作することはできませんが、cygwin に含まれる cygwin1.dll を同一フォルダにおくことで、Cygwin なしでも Windows のプロンプトで動作することができます

～～～～～～～～～～～～～～～セーフモードについて～～～～～～～～～～～～～～～～～～～～～～
Sharp 製 IS05 には、セーフモード起動機能があります。電源オン時にシャッターボタンを半押しにします。

すると、セーフモードの文字が表示されて起動します。通常時と異なるのは、後からインストールしたアプリは一切認識されず、起動もされない、できない状態で立ち上がります。これは、後からｲﾝｽﾛｰﾙしたアプリが原因で立ち上がらなくなるような場合に利用して、そのアプリを削除したりすることができるようです。

ただし、再起動すると、ホーム画面に設定したショートカットが消えていたりするので、再設定が必要にる場合があるようです。

このセーフモードと、リカバリーモード起動とは、関係ないように思われます。

～～～

8. その他のファイルシステムイメージ

Boot、Recovery 領域以外のフラッシュデバイス区域について見てみます。

1）区画名”system”は、文字通り LinuxOS や AndroidOS を含めたシステムのファイルが入っています。

この領域は、YAFFS2 のファイルシステムで格納されており、直接ドライブとしてマウントすることができるようになっています。

実際に通常起動時のブート領域の情報の初期化処理(init.rc)をみてみると

mount yaffs2 mtd@system /system ro

読み取り専用にマウントされていることがわかります。そのフォルダは所有者は root ですが、

drwxr-xr-x root root 2011-07-01 00:00 system

と一般ユーザーもその中を読んだり、実行することは許されています。

いくつかのフォルダは読めない部分もあるが、とりあえず中身を覗いてみると、

```
$ls -l /stsem
drwxr-xr-x root     root          2011-07-01 00:00 app          :本体内蔵アプリ郡
drwxr-xr-x root     root          2011-07-01 00:00 asg          :本体アプリの一部の asg というデータ
drwxr-xr-x root     shell         2011-07-01 00:00 bin          :Linux のコマンド郡
drwxr-xr-x root     root          2011-07-01 00:00 data         :dat ファイル？
drwxr-xr-x root     root          2011-07-01 00:00 dictionary   :日本語、和英・英和辞書
drwxr-xr-x root     root          2011-07-01 00:00 etc          :システムのもろもろの処理郡
drwxr-xr-x root     root          2011-07-01 00:00 fonts        :内蔵フォント
drwxr-xr-x root     root          2011-07-01 00:00 framework    :Andoroid のフレームワーク
drwxr-xr-x root     root          2011-07-01 00:00 lib          :ライブラリ郡
drwxr-xr-x root     root          2011-07-01 00:00 media   :アラーム・サウンド、画像関連データ
drwxr-xr-x root     root          2011-07-01 00:00 pubkey
drwxr-xr-x root     root          2011-07-01 00:00 sb
drwxr-xr-x root     root          2011-07-01 00:00 smotion   :ホーム画面関連画像
drwxr-xr-x root     root          2011-07-01 00:00 usr
drwxr-xr-x root     shell         2011-07-01 00:00 xbin
-rw-r--r-- root     root       3361 2011-07-01 00:00 build.prop
-rw-r--r-- root     root        649 2011-07-01 00:00 ts.conf
-rwxr-xr-x root     root      22528 2011-07-01 00:00 padding0.pad
-rwxr-xr-x root     root     188416 2011-07-01 00:00 padding1.pad
                                                    ・
                                                    ・
                                                paddingxxx.pad というファイルが続く・・
drwx------ root     root          2011-12-16 00:38 lost+found
```

といった情報が格納されており、これらの情報は通常運用上、書き換えは行われない。（読み取り専用）
これらのシステムがデータを書き飲む場合は/data フォルダを利用することになります。

2）区画名”userdata”は、システムで書き込むデータや、後からインストールするアプリケーションを格納する領域となっています。アプリケーションのデータもこの中に入ります。

この領域も、YAFFS2 のファイルシステムで、

mount yaffs2 mtd@userdata /data nosuid nodev

とマウントされています。この領域は所有者による権限設定がなされているため、System 以外のユーザにはその中身は読み書きすることはできません。

drwxrwx--x system system 2011-11-09 18:06 data

そこで、ルート権限を取得してその中身を覗いてみました。

すると/data の下には、複数のディレクトリが存在します

```
#ls -l /data
drwxrwxrwx root     root            2011-11-08 10:38 media       :SD カードがない場合のドキュメント格納
drwx------ system   system          2011-12-13 01:13 backup      :バックアップシステム用？
drwx------ system   system          2011-11-02 18:00 secure      :Secure 領域
drwxrwxr-x system   system          2011-11-23 21:55 anr         :トレース情報
drwx--x--x root     system          2011-11-26 11:09 tmp         :コマンドログ情報
drwxrwxrwx root     sdcard_rw       2011-12-13 01:13 synergy     :何らかのデータベース？
drwxrwxrwx root     root            2011-11-02 18:00 shared      :？（空だった）
drwxrwx--- root     qcom_oncrpc     2011-11-02 18:00 gpsone_d    :？（空だった）
drwxrwxrwx root     qcom_oncrpc     2011-11-02 18:00 wiper       :？（空だった）
drwxrwxrwx shell    root            2011-11-02 18:00 wpstiles    :？（空だった）
drwxrwxrwx root     root            2011-11-02 18:00 fota        :？（空だった）
drwxrwxrwx root     root            2011-11-21 10:22 fonts       :ユーザーフォント格納場所（空）
drwxrwx--- system   cache           2011-12-13 00:58 cache       :キャッシュ？（空だった）
drwx------ root     root            2011-12-13 01:14 sb          :？（空だった）
drwxrwx--x system   system          2011-12-13 01:13 dalvik-cache  :Java 実行環境のキャッシュ
drwxrwx--- radio    radio           2011-12-13 01:13 radio       :
drwx------ root     root            2011-12-16 00:32 property    :システム情報？
drwxrwx--x system   system          2011-12-12 12:30 app         :ダウンドードアプリ（apk）格納場所
drwxrwx--x system   system          2011-11-02 19:39 app-private  :au-wifi アプリが入った
drwxrwx--x system   system          2011-12-02 08:35 data        :各アプリケーションのデータエリア
drwxrwx--x shell    shell           2011-11-25 02:18 local       :adb シェルでアクセス可能な領域
drwxrwx--t system   misc            2011-11-02 19:50 misc        :デバイスシステムの情報格納
drwxr-x--- root     log             2011-11-02 18:00 dontpanic
drwxrwx--- root     root            2011-11-02 18:00 lost+found
drwxrwxr-x system   system          2011-12-16 00:32 system      :システムのワーク領域（OS が使う）
```

となっています。

media フォルダは、SD カードの PRIVATE、SD_VIDEO フォルダ同等の役割として確保されていたことがわかります。

そしてほとんどの情報のメインが、/data/data/フォルダで、ここにシステム領域にあるアプリや、ダウンロードしたアプリを含めて、アプリ自身が管理できる領域となっています。そしてこのフォルダの下に、各アプリごとのフォルダが生成されています。

```
#ls -l /data/data
drwxr-x--x netapplication netapplication        2011-11-02 18:07 com.kddi.android.auoneidsetting
drwxr-x--x app_63   app_63          2011-11-02 18:00 com.android.bluetooth
drwxr-x--x app_62   app_62          2011-11-02 18:00 jp.co.sharp.android.vbookmark
drwxr-x--x app_61   app_61          2011-12-10 02:38 com.android.browser
drwxr-x--x app_92   app_92          2011-12-13 01:13 com.adobe.flashplayer
drwxr-x--x aqq_1008 aqq_1008        2011-11-02 18:00 jp.co.sharp.android.calc
drwxr-x--x app_60   app_60          2011-11-02 18:13 com.android.calendar
drwxr-x--x app_91   app_91          2011-12-13 01:13 jackpal.androidterm
drwxr-x--x app_59   app_59          2011-11-02 18:00 jp.co.sharp.android.calendarselector
drwxr-x--x app_58   app_58          2011-11-04 01:44 com.android.camera
drwxr-x--x app_57   app_57          2011-11-02 18:00 com.android.certinstaller
 ・・・・・・・など多数
```

のように、各アプリごとに付与されたユーザ／グループ ID でのみ読み書き可能なデータフォルダとなっており、その他のユーザーからはアクセスできないようになっています。

各アプリのキャッシュデータもこの領域に生成されています。

しかし、こうして見てみると、意外と一般ユーザがアクセスできる領域も存在していることがわかります。

3）区画名”cache”はキャッシュ情報用のフォルダ、その他データベース用として、”persist”、”battlog”、”ldb”、”calllog”といったフォルダがマウントされています。

```
mount yaffs2 mtd@persist /persist nosuid nodev
mount yaffs2 mtd@cache /cache nosuid nodev
mount yaffs2 mtd@persist /persist nosuid nodev
mount yaffs2 mtd@battlog /battlog nosuid nodev
mount yaffs2 mtd@ldb /ldb nosuid nodev
mount yaffs2 mtd@calllog /calllog nosuid nodev
```

これらのフォルダの権限は以下のように制限されています。

```
drwxrwx--x system   system          1980-01-06 09:00 persist
drwxrwx--- system   cache           2011-11-02 17:59 cache
drwx-----x root     root            2011-10-20 11:42 battlog
drwxrwx--x app_10   app_10          2011-11-16 21:47 ldb
drwxrwx--x system   system          2011-11-02 18:00 calllog
```

いずれも一般ユーザには見えません。

最近話題となっている、LifeLogServise（jp.co.sharp.android.lifelog.database）というのが、ユーザの操作（メールや電話、Web アクセス先、アプリ操作など）を記録しているといういことで、プライバシー問題として騒がれています。

そのデータは、/ldb という領域に、1 つのファイルシステムを使って記録されているようである。

不具合が生じたときのそのときの操作などを、メーカーが追いかけるときに利用されるのであろうが、このデータが通常時に、ネットワークを介して勝手に送信されていることはなさそうです。

9. 一時ファイル用フォルダ

システムやアプリが一時的に利用するデータを格納する領域として、RAMDISK（tmpfs）が用意されています。 そして、/mnt/asec 、/mnt/obb、/sdservice_tmp、/bkrs_data_tmp、 /obex_data_tmp といったフォルダにマウントされています。

```
mount tmpfs tmpfs /mnt/asec mode=0755,gid=1000
mount tmpfs tmpfs /mnt/obb mode=0755,gid=1000
mount tmpfs tmpfs /sdservice_tmp nosuid nodev
mount tmpfs tmpfs /bkrs_data_tmp nosuid nodev
mount tmpfs tmpfs /obex_data_tmp nosuid nodev
```

この tmpfs はトータル領域として決まったサイズ（たとえば165 MB）が確保されており、それぞれのフォルダが165 M 使用可能な RAMDISK に見えますが、各フォルダで同じ tmpfs の資源を共有している形になります。（ただしそれぞれの領域は排他的に見え、データが混在することはありません）

これらの領域のフォルダも、一般ユーザのアクセス権は制限されています

```
drwxrwxr-t root     system          2011-11-16 19:41 obex_data_tmp
drwxrwxr-t app_13   app_13          2011-11-16 19:41 bkrs_data_tmp
drwxrwxr-t root     root            2011-11-16 19:41 sdservice_tmp
```

r-t の t は所有者以外の削除ができないフォルダ設定(Sticky Bit)であることを示しています

10. SD カードのマウント

SD カードも/mnt/sdcard にマウントされています。

```
mount   /mnt/sdcard vfat   rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,
          fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,iocharset=iso8859-1,
            shortname=mixed,utf8,errors=remount-ro 0 0
```

のようにマウントされ、noexec とあるため実行可能なファイルの実行ができなくなっています。

すべてのマウント情報は mount コマンドで確認できます（設定確認には Root 権限は不要）

```
 $ mount
mount
rootfs / rootfs ro,relatime 0 0
tmpfs /dev tmpfs rw,relatime,mode=755 0 0
devpts /dev/pts devpts rw,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,relatime 0 0
tmpfs /synthesis tmpfs rw,nosuid,nodev,relatime 0 0
tmpfs /obex_data_tmp tmpfs rw,nosuid,nodev,relatime 0 0
tmpfs /bkrs_data_tmp tmpfs rw,nosuid,nodev,relatime 0 0
tmpfs /sdservice_tmp tmpfs rw,nosuid,nodev,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
tmpfs /mnt/asec tmpfs rw,relatime,mode=755,gid=1000 0 0
tmpfs /mnt/obb tmpfs rw,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/mtdblock5 /system yaffs2 ro,relatime 0 0
/dev/block/mtdblock11 /data yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock6 /persist yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock1 /cache yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock8 /battlog yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock10 /ldb yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock9 /calllog yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/vold/253:0 /mnt/sdcard vfat rw,dirsync,nosuid,nodev,noexec,relatime,u
id=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,iocharset
=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
/dev/block/vold/253:0 /mnt/secure/asec vfat rw,dirsync,nosuid,nodev,noexec,relat
ime,uid=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,ioch
arset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
tmpfs /mnt/sdcard/.android_secure tmpfs ro,relatime,size=0k,mode=000 0 0
fatsh /cprm fuse.fatsh rw,nosuid,nodev,noexec,relatime,user_id=0,group_id=0,defa
ult_permissions,allow_other 0 0
$
```

Sd カードはのフォルダは、さらにルートの/sdcard へリンクされているため、SD カードもルートからのフォルダ/sdcard としてアクセスすることができます。

なお、SD カードに、セキュア対応データを格納する場所もあり、ワンセグの録画情報、コピー不可能な音楽データなど、セキュア専用のフォルダが別途マウントされています。

マウントされたフォルダ毎のサイズ、使用量(残量)を確認するには、df コマンドを使います。

```
$ df
df
Filesystem              Size   Used   Free   Blksize    mountsource
/dev                    164M   136K   163M   4096       tmpfs
/synthesis              164M   216K   163M   4096       tmpfs
/obex_data_tmp          164M     0K   164M   4096       tmpfs
/bkrs_data_tmp          164M     0K   164M   4096       tmpfs
/sdservice_tmp          164M     0K   164M   4096       tmpfs
/mnt/asec               164M     0K   164M   4096       tmpfs
/mnt/obb                164M     0K   164M   4096       tmpfs
/system                 410M   387M    22M   4096       /dev/block/mtdblock5
/data                   421M    96M   325M   4096       /dev/block/mtdblock11
/persist                  3M   772K     2M   4096       /dev/block/mtdblock6
/cache                   30M     1M    28M   4096       /dev/block/mtdblock1
/battlog                  3M     1M     1M   4096       /dev/block/mtdblock8
/ldb                     18M     1M    16M   4096       /dev/block/mtdblock10
/calllog                  2M   828K     1M   4096       /dev/block/mtdblock9
/mnt/sdcard              14G     3G    11G   32768      /dev/block/vold/253:0
/mnt/secure/asec: Permission denied                     /dev/block/vold/253:0
/cprm: Operation not permitted                          fatsh
```

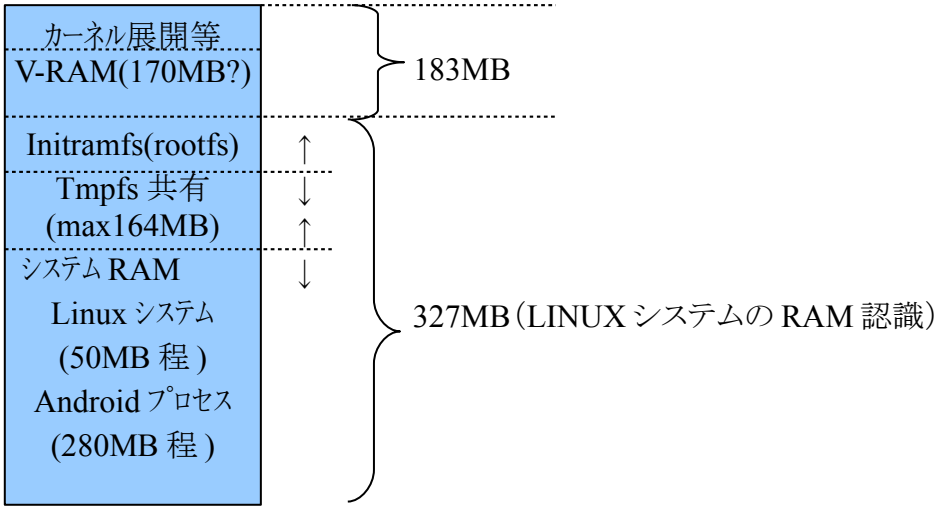
tpmfs は 164M の領域をいくつかのフォルダで共有していています。それぞれに 164M あるわけではありません。

11. RAM デバイスについて

システムとしてのメモリ状況を確認するには、/proc/meminfo を見ます。

```
$ cat meminfo
cat meminfo
MemTotal:       335960 kB
MemFree:         35132 kB
Buffers:          5072 kB
Cached:          50740 kB
SwapCached:          0 kB
Active:          96472 kB
Inactive:       160436 kB
 ・  ・（省略）・  ・
```

と、LINUX システムとしては、335960kB(327MB)を認識しているようです。ということは、大体 512MB 位が RAM デバイスの容量となるのかな？と推測できます。



この差分、188328kB(183MB)は、初期カーネル展開用、VRAM 領域（画面表示用）等が含まれていると考えられます。

フラッシュデバイスのブート領域の展開情報から、カーネル情報は、8MB くらい、initramfs は、300KB（Boot）から 500KB（Recovery）位の展開量であることがわかります。

VRAM が、約 170MB とは多いような気がしますが、IS05 は解像度が、854×480 ドット、27 万色（18bit）をサポートしています。

ちなみに、他機種比較情報として、IS03 の VRAM が 126MB 程度という情報もあり、IS03 が 960x640 の 65000 色であることから、そんなものかもしれません。

initramfs や、Tmpfs は、決まった容量が確保されているわけではなく、必要な分だけメモリをとるようになっています。使用しない分はシステムの領域に割り当てられます。Tmpfs は制限値として 164M となっています。

一方、携帯本体のメニューで、アプリケーションの管理で実行中のサービルスを見ると、
使用中　　:144MB（実行中サービスの使用量）
キャッシュしたプロセス:約 80MB（リストの各アプリの使用量合計）
空き　　:131MB

のようになっています。

Android はアプリを終了してもキャッシュとしてメモリ上に残しておき、メモリが不足する削除する仕組みとなっているため、完全にフリーなメモリは少ない状態である場合があります。

上記の場合使用中144 MB に対して、キャッシュしたプロセス（リストのサイズを合計したもの）が、約80 MB 存在しました。そして空きメモリが133 MB のようです。

空きメモリとはキャッシュしたプロセスのサイズと、未使用のサイズの合計のサイズを表しているようです。
（キャッシュしたプロセスを全消去しても空きメモリ表示は増加しなかった）

大体使用中＋空きメモリが、260 から 280MB くらいに見えます。

LINUX のシステム認識メモリが 335MB に対して、Android プロセスとしては、280MB 程となっているのでしょう。
50 MB ほどはシステムの利用分や tmpfs の増加を考えたときの猶予を持たせているのかもしれません。

先ほどの、メモリ情報をみなおすと、335860KB でフリーが 35132KB しかないのは、キャッシュしたプロセスが常に存在するからで、システムの Chashed が 59749KB だから、Android のプロセス利用分合わせて 338M B ということでほぼシステム認識のサイズになっているようにみえます。

12. システムの起動の流れ

以上のように、ルートのブート領域から展開したファイルシステムをベースに、フラッシュデバイスのファイルシステムや RAMDisk などをフォルダとしてマウントしていくことでファイるシステム全体を構築していくようになっています。実際はこれら以外のフォルダのマウント等こまごまな追加もあります。

そして、ルートの initranfs も最後にリードオンリーに再マウントされるようになっており、一般ユーザが、アクセスできるのは、SD カードと、/data フォルダの本の一部となっていることがわかります。

さてファイルシステムの構成が見えたところで、そのようにシステムが起動しているのか、を見るために、
psコマンドを実行してみます。すると、

```
$ ps
ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     456    332   ffffffff 00000000 S /init
root      2     0     0      0     ffffffff 00000000 S kthreadd
root      3     2     0      0     ffffffff 00000000 S ksoftirqd/0
root      4     2     0      0     ffffffff 00000000 S events/0
root      5     2     0      0     ffffffff 00000000 S khelper
                .       .          .        .
                .       .          .        .
system    111   1     956    232   ffffffff 00000000 S /system/bin/servicemanager
root      112   1     5568   484   ffffffff 00000000 S /system/bin/vold
root      113   1     8364   1048  ffffffff 00000000 S /system/bin/usbmgrd
root      114   1     3996   364   ffffffff 00000000 S /system/bin/netd
root      115   1     812    200   ffffffff 00000000 S /system/bin/debuggerd
root      116   1     102020 22868 ffffffff 00000000 S zygote
media     117   1     62660  6784  ffffffff 00000000 S /system/bin/mediaserver
bluetooth 119   1     1460   596   ffffffff 00000000 S /system/bin/dbus-daemon
                .       .          .        .
                .       .          .        .
system    348   116   285292 47724 ffffffff 00000000 S system_server
system    441   116   128412 31356 ffffffff 00000000 S com.android.systemui
app_27    449   116   161756 25636 ffffffff 00000000 S jp.co.sharp.android.wallpaper3d:drawpaper
app_24    450   116   150024 35420 ffffffff 00000000 S jp.co.omronsoft.iwnnime.ml
radio     464   116   162504 33852 ffffffff 00000000 S com.android.phone
system    465   116   111096 19848 ffffffff 00000000 S com.kddi.android.fota
                .       .          .        .
                .       .          .        .
app_91    5685  116   126264 27360 ffffffff 00000000 S jackpal.androidterm
app_91    5781  5685  880    352   ffffffff 00000000 S /system/bin/sh
shell     5785  5781  1032   348   00000000 afd0aabc R ps
  $
```

といった具合に、起動プロセスが表示されます。

ブート領域から起動したカーネル処理は、RAMDISK初期展開を行うと、ルート権限でプロセスID0として、/init というプログラムを実行しています。このプログラムが、最初の起動となり、init.rc に記述された処理を実行していきます。その中で上記ファイルシステムのマウント等をおこなっていることがわかります。

そして、プロセス ID 0の init プログラムは、プロセス ID1 の kthreadd というスレッド管理プログラムを起動し、この 2 つの処理（ID0,1）が、さまざまな LINUX のサービスプログラムを起動していきます。

そして、rootからの初めてのユーザが、systemでservismanagerを起動しています。

さらにsystemは、system_serverを立ち上げ、直後にcom.android.systemuiを起動しています。
このあたりから、AndroidOSの起動が始まっていることがわかります。

そして、一般ユーザである、Terminal Emulator(jacpal.androidterm)がapp_91というユーザで起動して、システムのシェル(/systen/bin/sh)を起動、そして、psコマンドを発行（実行）している、ということになります。

なおリストの、PIDがプロセスID、PPIDが親IDとなっており、すべてがID0につながっています。

このようにブート領域のカーネルから立ち上がったLinuxシステムがAndroidのシステムを起動しているということがわかったと思います。

さて、このようなシステムを垣間見れたわけですが、一般ユーザ権限では、制限がされており、みることができない部分や書き換えできないようになっています。これは携帯電話システムとしての安全性と、信頼性を守るために、メーカーがプロテクトしていることになります。

携帯電話として、一般ユーザーの自由度があがると、システム自体かわってしまうので、メーカーはそういったところまでのアクセスはできないように、なっているのが通常である。

ところが、セキュリティの盲点をついて、ルートユーザーになることで、これらの制限が解除されることもしられています、

この後は、Rootのユーザー権の取得について、見ていきます。

〜〜〜〜〜〜〜〜〜〜標準ブラウザでSDカード内のファイルを直接アクセス〜〜〜〜〜〜〜〜

SDカードに保存した、HTMLファイルをブラウザで直接アクセスする方法があります。

ブラウザを立ち上げて、アドレスの指定に以下のように指定します。

content://com.android.htmlfileprovider/sdcard/フォルダ名/ファイル名(.html等)

WEBページ公開前のファイルをチェックするのに利用可能です。また、JavaScriptで書いたプログラムの実行もできるので、ある意味プログラムとして、アプリの開発が可能です。

ちょっとした、複雑な計算をさせるときとか、特別なアプリも必要なく利用できるので便利かもしれません。

〜〜

13. ルートのユーザー権の取得について

携帯のシェルにアクセスした場合、Terminal アプリのユーザー ID でログインした形、adb shell だと Shell ユーザーでログインされた形で、シェルに入ることはすでに述べたとおりですが、
この場合、権限のないフォルダやファイルのアクセスができないことになります。

そこで、すべての権限でアクセスしたい場合、ROOT ユーザーになることが必要となります。

通常の Linux で Root に入るためには、Linux コマンドシェルでsuコマンドをつかいます。このコマンドは、任意のユーザーにログインするもので、指定がない場合、Rootに指定されます。

```
$su
password:
```

と表示され、パスワードを入力して一致した場合ルートに入ります。正しいパスワードを認識すると、

```
#
```

とプロンプトが表示されます。

ところが、携帯には、su プログラムは入っていません。

一般に、ルートを取得するのに、ルート権限で動作するプログラムというのが考えられます。
よく例えで用いられるのに、パスワードの情報へのアクセスがあります。

パスワード情報は、極秘情報なので、Root 権限で作られたファイルに格納されていることになっています。
ところが、一般ユーザーが、自分のパスワードを変更したりした場合、一般ユーザ権限ではパスワードファイルの更新ができません。

そこで、一般ユーザでも、Root 権限でパスワードファイルを更新するプログラムを一般ユーザが起動することでその Root 権限のファイルを更新してもらいます。

ではその Root 権限で実行するための仕組みとして、suid、guig という物があります。

これは、ファイルの権限に、追加することで、そのファイルのユーザ、グループ以外でも、そのファイルのユーザー、グループの権限でアクセスできるという特別な権限です。

たとえば、
-rwx------x root root xxxxx 2011-12-12 exec
という実行ファイルがあったとします。これは一般ユーザが実行可能だとしても、一般ユーザ権限で起動されるので、この実行プロセスは、root 権限のパスワードファイルにアクセスすることはできません。

そこで suid という属性を付与すると、
-rws-----x root root xxxxx 2011-12-12 exec
と所有者権限の”x”の部分が”s”に変わります。

すると、このファイルは、他のユーザーであっても、所有者の権限でを実行することができ、つまり root 権限で、root 所有のパスワードファイルを書き換えることができます。

同じように、guid は、ファイルの所有グループの権限で実行できます
例：-rwxrws--x root system xxxxx 2011-12-12 exec

だと、一般ユーザが、所有グループの system のグループ権限で実行することになり、この実行プロセスは、system グループがアクセス可能なファイルを更新することができます。

このように、一般ユーザが所有ユーザに成り代わって実行する権限を与えることができるとすると、

```
$ ls -l /system/bin/sh
ls -l /system/bin/sh
-rwxr-xr-x root     shell       82840 2011-07-01 00:00 sh
$
```

もしこのファイルが

```
-rwsr-xr-x root     shell       82840 2011-07-01 00:00 sh
```

となっていたら、 Root ユーザの所有している実行ファイルの、シェルプログラムに、suid が付与されていれば、一般ユーザはいつでもルート権限でシェルを起動できるはず。ということになります。

このような手段を用いて、ルートに入る抜け道をつくることで Root に入る手段が作られたりしました。

ただし、携帯電話のシステムで考えると、システム領域は書き換えできないしコピーしてきても、実行できるところは、/data くらいです。しかしじつは/data フォルダはマウント時に suid の実行が許可されていません。
マウント情報をよく見ると、nosuid と記載されています。これはほとんどのマウントでそうなっています。
なっていないのは、書き換えできない/system か読みより専用にリマウントされたルートファイルシステム部くらいです。
しかも、一般ユーザ権限でコピーしても、ファイル所有者は一般所有者に格下げされるし、そもそも、suid を付与できるのは、Root 権限のみとなっています。

要するに、一旦 Root 所有者になって、このような実行ファイルと実行可能なフォルダをマウントさせないとできない。。という矛盾になり、通常は結局ルート権限にはなれない、ということになります。

～～～～～～～～～～～～～～～～ファイルの属性変更～～～～～～～～～～～～～～～～～～～
ファイル（フォルダ）の属性を変更するには、chmod コマンドを使います。
属性の rwxrwxrwx の rwx を数値で表し、r=4,w=2,x=1 の合計数の 3 桁の数値であらわし、

```
$chmod 755 filename
```

のように表して設定します。（755=rwxr-xr-x）
suid、guig を付与するためにさらに数値をつけて、suid=4、guid=2、stickybit=1 で追加します。

```
$chmod 6755 fileneme
```

この場合、6755 = rwsr-sr-x を意味します。

stickbit は削除権を所有者のみにする設定で、その他のユーザの実行権表示部が t になります。
（例：rwxr-xr-t）
～～～

ところが、セキュリティの脆弱性をついたプログラムにより、一時的に Root になることを見つけた人がいます

Sharp 製の場合、http://twitter.com/goroh_kun （2011.11.19 記事）という Twitter ページでも公開されていますが、脆弱性をついて実行権限を root にするという、breaksuidprox（近接センサの脆弱性を突いたもの）、breaksuidhsdiag（ハンドセットの脆弱性を突いたもの）というのがあります。

これらを実行すると、その時点から起動するプロセスは Root ユーザになってしまします。
（IS05 ではどちらも有効でしたが機種によっては効果のない場合もあります）
ちなみに、これ以降の操作はシステムを破壊する可能性もありますので、自己責任となります。
またメーカーのサポートも受けられなくなる可能性もありますので、十分注意してください。

現在、Android2.3.4 にバージョンアップされてからこの方法は無効となりました。ただし他の方法で最新版でも Root が取る方法があるらしいです。

＊すべて起動するプログラムが Root 権限になるという特殊な状態なので、あとで削除できないファイルが生成されたりする可能性もあり、不安定な状態といえます。

では、とりあえず、adb でこのファイルを送り込んで実行権を与えます。

```
>adb push breaksuidprox /data/local/tmp
>adb shell chmod 755 /data/local/tmp/breaksuidprox
```

そして、実行してみます。

```
>adb shell
$ /data/local/tmp/breaksuidprox
```

メッセージがでて終了するが何も変わらず。。。

携帯の TerminalEmulator を実行すると、

```
$
```

そこで、携帯の設定のアプリケーションの管理から、TerminalEmulator のプロセスを終了させます。

そして、再度起動すると、

```
#
```

になります。

```
#id
  uid=0(root) gid=0(root) group=1015(sdcard_rw),3003(inet)
```

のように、ルートユーザ権限になっていることがわかります。

最初なぜ変わっていなかったかというと、Android のアプリは終了しても、待機状態でプロセスが継続して

いるため、新たに起動しなおす必要があったためです。（起動していなかった場合は即＃表示されます）

さてこれでルート権限で、携帯本体からは、制限された情報にアクセスできるわけですが、

一方、adb shell の起動は何度やっても shell ユーザーにしかなりません。adb shell はもともとシステムが用意した shell ユーザーで起動するようになっているため、adb shell を再起動しただけではルートユーザになりません。

そこで先ほど、出てきた Root 権限で起動するプログラム、というのが必要になってきます。

とりあえず携帯本体で ROOT ユーザになったので、そこで Root 権限のファイル操作ができます。

実際に、suid プログラムを ROOT 権限で作って、実行できる環境を用意する必要があります。

そのため suid のプログラムを動かすことができる、ルートフォルダを利用します。そこで、ルートフォルダを読み書き可能なフォルダにリマウントします。(mount 変更は root 権限必要）

```
#mount -o rw,remount,suid rootfs /
```

すると、システムパスがととっている/sbin/に実行ファイルをいれることで、suid 許可プログラムを実行できる用になります。

ところで、実際のところ、さきの sh コマンドをコピーして suid を付与したものでは、id が変わりませんでした。Linux には sh に suid の対策がされているということのようです。

そこで、Android ソースの su プログラムを、取得してくることにします。（上記 Root 取得前に行っておきます）

Android ソースの/system/extras/su に su.c という C プログラムが見つかります。
/system/core/include/private にある android_filesystem_config.h も必要になるので取得します。

そして Cygwin の任意のユーザーフォルダ（/home/（user）/su/jni）で AndroidNDK でコンパイルします。
（インクルードファイルは/home/(user)/su/jni/private/に入れます）
そして同じく su.c ファイルと同じフォルダにあった Android.mk ファイルを/home/（user）/su/jni に入れます。

そして、cygwin を起動して AndroidNDK でコンパイルを起動します

```
$cd /home/（user）/su
$ndk-build
```

lib フォルダにできた su 実行ファイルを、sush に名前を変えておきます。
これは、携帯システムに su プログラムが存在するかのチェックを行っているらしく、検出されると一部の機能が動作しないようになっているということらしいです。

とにかくこの sush を携帯のフォルダ/data/local/tmp に転送しておきます。(Root 取得前に転送しておきます)

```
>adb shell sush /data/local/tmp
```

さてここで、携帯で Root で取得したシェルにおいて、

```
#cd /data/local/tmp
#chown root sush
#chmod 6755 sush
```

を実行して、所有者を Root、実行権と suid の付与を行います。

```
#is -l sush
-rws-r-sr-x  root   shell    xxxxx  2011-12-12 xx:xx  sush
```

となります。
そして、このファイルを、リマウントされた/sbin にコピーします。

```
#cp sush /sbin
```

(cp は busybox を展開しておきます)

これで、adb shell から sush が起動できます。

```
>adb shell
$sush
#
```

となり、

```
#id
id
uid=0 gid=0 groups=1003(graphics),1004(input),1007(log),1009
(mount),1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net_bt),3003(inet)
#
```

のように Root 権限でシェルが起動していることがわかります。

実は、Android ソースの su はユーザー ID グループ ID の変更を行ってシェルを起動しているプログラムとなっていました。
実際の LINUX の su 機能からは簡略化されています。パスワードの処理がないのは、携帯の LINUX の Root や各プログラムに与えられたユーザごとにパスワード管理もしてられないため、パスワードなしで運用しているものと思われます。

ともかく、これで adb shell からも Root 権限でシェル操作ができるようになり、制限された情報も見ることができるようになります。

なお、/data/local/tmp フォルダは、携帯本体の Terminal からアクセスできるようにアクセス権を変更しておいたほうがよいでしょう。

```
>adb shell chmode 777 /data/local/tmp
```

14. Root 取得（一時取得）の運用例

Root ユーザになって何ができるか。。いくつかの例をあげておきます

1）既に、述べましたが、フラッシュデバイスの格納ブロックごとのデータを吸い上げることができます。
そして、編集して書き戻すことができます。書き戻しには NAND ロック解除が必要です。

2）プレインストールのアプリを起動しないようにすることができます。

```
#pm list packages -f
```

（これは一般ユーザ権限でも可能）

を実行すると、インストールアプリの一覧が出るので、そこから停止したいアプリ名を調べます。
そして、

```
#pm desable <アプリ名>
```

を実行します。

これは一般ユーザでも起動できますがすぐに立ち上がってしまいます。

3）ブラウザのキャッシュデータを SD カードに移動する。
ブラウザのキャッシュデータはすぐに大量にたまり、/Data フォルダ領域を圧迫します。
そこで、ブラウザのキャッシュ領域を SD カードに移すことができます。

```
#cd /data/data/com.android.browser/cache
#rm -R webviewCache
#mkdir /sdcard/webviewCache
#ln -s /sdcard/webviewCache webviewCache
```

ただ、実際 SD カードには4 MB までしか入らずそれを超えたら/data/領域が増えていったという
状況であった。

それ以外のアプリのキャッシュに関するところも含めて SD カードにリンク変更していけば、かなり
削減できるかもしれません

しかし、VFAT ファイルシステムの SD カードは余りもの書き換えが頻発すると故障の原因になるかも
しれません。フラッシュデバイスとしては、ファイルシステムとしては Yaffs2 の方が信頼性があります。

4）Root 権限が必要なアプリを起動する目的で Root 取得を行おうとする場合もあるとおもいます。
ところが、この Root 取得状態は先にものべたとおり不安定な状態であるためこの状態で通常運用は
避けたほうがよいでしょう。

この Root 権限でシェルに入れる、sush がルートフォルダ/sbin に常にあれば、最初の危険な一時 Root
取得プログラムを　実行しなくても、Root に入れることになります。

そうすると、シェルからプログラムを起動する方法がある（am コマンド）のでそこでアプリを起動したら
アプリ単位で Root 権限で起動することもできると思います。

```
#cd data/app/(アプリパッケージ名)
#am start -a android.intent.action.MAIN -n (アプリパッケージ名)/.（アクティビティ名）

-n 部は AndroidManifest.xml にある package/activity android:name
```

しかし、ルートフォルダは RAMDISK なので、ルートフォルダに作成したのファイルは電源を切ると、消えてしまいます。 当然一時 Root 取得権も消えてしまします。

うまくできているというか。。。。

そこで、ルートフォルダの RAMDISK 領域を書き換えるということをしたくなってくると思います。
つまりフラッシュデバイスのブート領域の書き換えです。

15. NAND ロックの解除

フラッシュデバイスのブート領域には、NAND ロックがかかっていて、通常書き換えはできないようになっています。

そこで、また、Root 一時取得プログラムを作った作者は、それを解除するプログラムを作られています。

詳細はそちらで見ていただいたほうがよいと思います。 http://twitter.com/goroh_kun (2011.11.23 記事)
ソースも公開されていますので参考にしてください。

nandunlockshspamp というプログラムで引数に、NAND のプロテクト情報の構造体アドレスを与えます。
NAND のプロテクト情報の構造体アドレスは、nandinfocalc というプログラムで算出するみたいです。
実行ファイルをダウンロードして格納した場所(/data/local/tmp)にて

```
#./nandinfocalc [boot /recovery]
```

算出アドレスを指定して

```
#./nandunlockshspamp 0xXXXXXXXX
```

その後、先にものべた、flash_image コマンドを実行します。

```
#./flash_image [boot/recovery]  書き込みイメージファイル
```

フラッシュデバイスの書き戻しにはこの NAND ロック解除が必要になるようです。
ただし、System は解除していないそうです。特に SYSTEM の書き換えが必要になることはないと思います。

ブートイメージの ramdisk 領域のルートファイル構成を変更すれば、初期処理(init.rc)を変更して、初期構成をかえることができるからです。

/system や/data を SD カードにマウントしてコピーして動作させたりすることも可能となります。
このとき SD カードはあらかじめパーテーションを切って、ext3/4 のファイルシステムにフォーマットします。

また、init.rc から/data フォルダ内のスクリプトファイルを呼び出すようにだけしておき、
あとから Root 権限での初期処理を追加変更できるようにしている方もいます。これだと、
/data フォルダのファイルでルートフォルダのリマウントや、root 起動プログラム sush を/sbin に転送しておくことでいつでも Root ログインできるようになります。
システム自体は通常なので Root 処理はそのログインシェルだけとなり、システムは安定した状態にできることになります。

/init.rc の rootfs を ro(リードオンリー)にリマウントしている記述をやめる(rw のまま)だけでも、拡張性が大きく上がると思います。

ただし、あくまでも自己責任で行ってください。書き換えを行うとメーカーサポート対象外になります。

以上で、携帯スマートフォンの中身を見ていくとともに、開発環境の構築、ツールの作成、Android ソースの取得、Root 権限の取得、といったことを、WindowsPC を使って調査し、その手段を見ていきました。

こういった情報は、いろいろなところに散在していて、情報がまとまっていないというのが現状です。

Sharp の携帯で見てみたたわけですが、基本的には同じような仕組みなので、このようにまとめたことで、その全体を理解するのに役立てればいいかと思います。

(2011.12.21 jun!)