「MeeGo の Android プラットフォームへの移植」

天野光隆
ミラクル・リナックス株式会社
E-mail: mamano@miraclelinux.com

概要　Android 端末は Android が動作するために設計されており、Android を使えばターゲットデバイスのすべての周辺機器を動かすことができるよう提供されている。しかし、Android は日に日に進化しているため、古いバージョンでリリースされた端末は、メーカーがアップデートを提供しない限りそのまま過去の遺物となってしまう。そこで、MeeGo を始めとした専用デバイスに特化した Linux ディストリビューションを移植することで、こうした「遺物」となりかけたデバイスに新たな可能性を発見することができるかもしれない。本論文では、Android プラットフォームの特徴と移植方法、MeeGo を実際に動かす例ついて述べる。

1. はじめに

Android プラットフォームは開発用のボードからエンドユーザ向けの製品まで、様々な形で販売されている。最近では最もユーザーの身近にあるのはスマートフォンだろう。もともと Android は、Linux カーネルをベースにスマートフォン向けに最適化された OS 環境だが、現在は Netbook やタブレットにも利用されており、今後は様々な機器で採用されていくと見られている。。

Android は現在も活発に開発が進められており、新しい機能を盛り込んだ新バージョンが早いサイクルでリリースされている。発売後の Android 搭載機が、そうしたバージョンアップにどこまで追随するかは、ひとえにメーカーの対応次第となっている。メーカーに新しいバージョンの Android をサポートする計画がないと、そのデバイスにおける今後のアップデートは期待できない。Andoroid 搭載機器が続々と発売される一方で、新しいバージョンへの対応が行われず置き去りにされる端末は、今後ますます増えることになる。

2. Android 端末の特徴

多くの Android 端末ではフラッシュメモリ上の OS イメージへのアクセスに MTD(Memory Technology Device)[i]という技術が使われている。例えば Google Nexus One の場合は、以下のように構成されている。順番や領域のサイズは Android 端末によって異なるが、フラッシュメモリから起動する Android はどれもこのように、name 列で列挙されているラベルで分割されている。

```
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 000e0000 00020000 "misc"
mtd1: 00500000 00020000 "recovery"
mtd2: 00280000 00020000 "boot"
mtd3: 09100000 00020000 "system"
mtd4: 05f00000 00020000 "cache"
mtd5: 0c440000 00020000 "userdata"
```

以下、これらの各領域について説明する。

mtd0(misc)は電源投入時に表示されるスプラッシュ画面などが保存されており、起動時にのみ使われる。Android 起動後はこの領域が使われることはない。

mtd1(recovery)は、復元領域として使われる。Android を復元する際はこの領域に保存されているリカバリ用のカーネルと initrd が用いられる。initrd 内にある init スクリプトによって SD カードへアクセスし、リカバリーファイルのイメージを展開、ルートファイルシステムへ書き込むといった流れで復元が行われる。

mtd2(boot)には、通常起動する際に用いられるカーネルが格納されている。通常の Linux でいうと/boot と同等の扱いといえる。この領域もカーネル(zImage)と initrd が書き込まれており、この initrd がルートファイルシステムとなる。initrd にある init スクリプトによって/sys や/proc、/dev などをマウントし mtd3(system)にある Android 本体の各種ファイルを実行するといった流れとなる。

mtd3(system)は Android が利用するすべてのコマンドや Android アプリケーション、必要なライブラリファイルが保存されている。mtd2(boot)からマウントされて利用することになる。実行ファイルやライブラリしかないため、リードオンリーでマウントされる。

mtd4(cache)は一時ファイルの保存先で、システムアップデート時のイメージファイルなどが置かれる。

mtd5(userdata)はユーザの電話帳やシステム設定、メール、ブラウザのアクセス履歴やキャッシュなどが保

存される。つまり再利用されるユーザーデータが保存される。なお、音声や動画、カメラで撮影した画像などのメディアファイルは mtd5(userdata)ではなく SD カードに保存される。

3. 移植方法

とにかくどのような方法もいいので、Android 以外の Linux を起動することが移植のファーストステップと言える。Android 端末で Linux を動かす方法はいくつかあるが、カーネルのロード方法とルートファイルシステムの置き場所に分けて説明する。まずはカーネルのロード方法から説明する。

方法 1: boot 領域を上書きし、通常起動する

書き込みツールを使って boot 領域を上書きし、電源を投入して自分のカーネルをロードする。前述の mtd 構成であれば mtd2 に相当する部分を上書きすることになる。オリジナルの Android カーネルが消去されるため、バックアップしておく必要がある。boot 領域のイメージは Android のツール「mkbootimg」から作ることができる。ソースコードは Android の git ツリー[ii]の mkbootimg から入手できる

方法 2: 書き込みツール経由で独自カーネルを転送

書き込みツール経由であれば、PC から自分のカーネルを転送し、一時的なカーネルとして起動が可能になる。フラッシュメモリ上には書き込まないため、Android システムには何も影響がない最も安全な方法である。ただし起動のたびに書き込みツールが必要になるため、PC がないと起動できない。安定して起動できる状態になるまでこの方法でテストを行うことになる。

方法 3: recovery 領域から復元モードを利用

書き込みツールを使って recovery 領域を上書きし、電源を投入して復元モードに切り替える際に自分のカーネルがロードされるようにする。前述の mtd 構成であれば mtd1 に相当する部分を上書きすることになる。Android の復元ができなくなるが、本体だけで Android とのデュアルブートを実現することができる。もちろん、前もってオリジナルの recovery 領域をバックアップしておけば Android の復元も可能になる。方法 1 と方法 2 の利点をあわせ持つので、もっとも最適な方法とも言える。

書き込みには fastboot と呼ばれるコマンドを使う。PC と Android 端末を USB ケーブルで接続し、ファイルの転送やフラッシュメモリの制御が可能である。fastboot を使えばフラッシュメモリ上の mtd 領域の消去やイメージファイルの書き込み、端末の再起動を行うことができ、Android G1、HTC 系スマートフォン (Google Nexus One, HTC Desire, HTC Hero 等)、その他 Qualcomm 社製 Snapdragon を搭載した端末であればほとんど利用可能だ。その他に fastboot を利用した nvflash(NVIDIA Firmware Update Utility)などがあり、NVIDIA Tegra が搭載された Android 端末ではこちらを使う。fastboot のソースコードは Android の git ツリー[iii]の fastboot を使う。Linux、Windows、MacOSX 用のソースもある。

3 つの方法を「図 1:カスタマイズカーネルの起動方法」で示す。方法 1, 3 は前述のとおり mkbootimg を使って boot 領域、recovery 領域に合わせたデータ構成を持つイメージを作り、initrd も用意する必要がある。方法 2 の場合、書き込みツールからカーネルパラメータ(起動時の引数)を指定できるので、initrd 不要である。

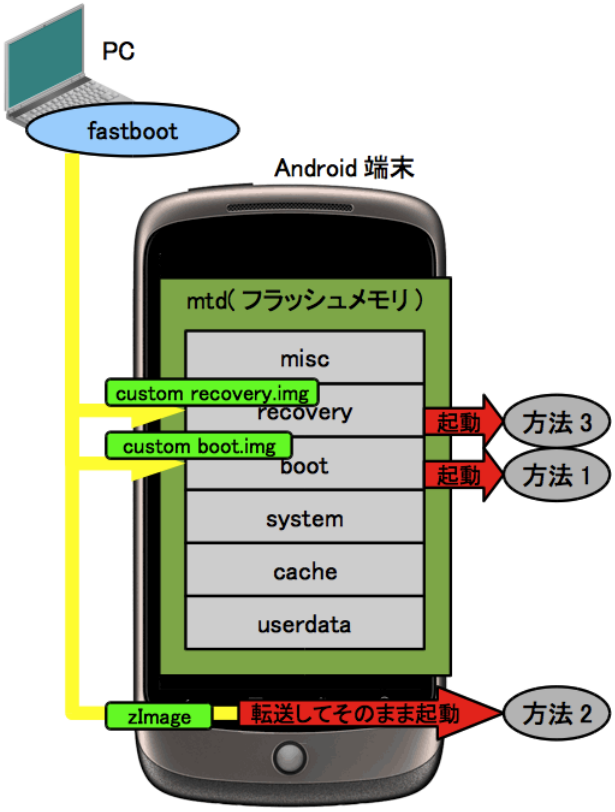

図 1:カスタマイズカーネルの起動方法

続いてルートファイルシステムの設置場所について解説する。

場所 1: system 領域を上書き

Android の場合、initrd と system 領域を結合してルートファイルシステムとしているが、通常の Linux であれば system 領域のみをルートファイルシステムとして使うことができる。initrd の init スクリプトで chroot を行えば、system 領域だけで完全な Linux 環境が構築できる。ここを上書きするとオリジナルの Android ファイルが消去されるため、バックアップしておく必要がある。フラッシュメモリの場合、ファイルシステムとして yaffs2 が使われているので、mkyaffs2 を使ってイメージを作成する。ソースコードは Android の git ツリー[iv]にある。Toshiba AC100(Dynabook AZ)では system 領域が ext3 なので、mkfs.ext3 で作成する。

場所 2: userdata 領域を上書き

userdata 領域は、system 領域よりもある程度大きめに確保されているので、より多くのファイルが保存できるルートファイルシステムとして利用できる。ただし上書きすると Android 上のユーザーデータがすべて消去される上、設定が保存できなくなるので Android 側は正常に操作できなくなる可能性がある。userdata 領域も system 領域と同様、yaffs2 でフォーマットされているので、mkyaffs2 でイメージを作成する。

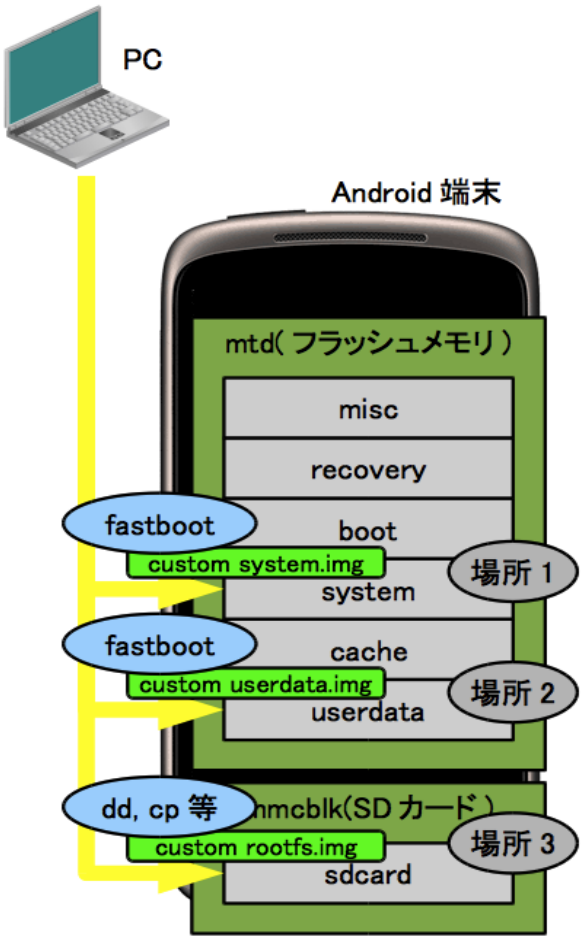

図 2:ルートファイルシステムの場所

場所 3: SD カードを利用

SD カードを挿入した状態で起動し、initrd から SD カードへ chroot を行えば、SD カードだけで完全な Linux 環境を構築できる。Android の system 領域を上書きせずに使えるため、デュアルブート環境として構築することが可能になる。SD カードに ext3 ファイルシステムなど自分のカーネルが利用可能なファイルシステムを構築し、必要なファイルを展開すればよい。

3 つの方法を「図 2:ルートファイルシステムの場所」で示す。system 領域や userdata 領域は fastboot を使えば転送できる。さらに initrd の init スクリプトや fastboot のカーネルパラメータをうまく使えばどの領域からでも起動することができる。SD カードを使えばフラッシュメモリよりも多くの容量を確保できるので、解析を行うには最も適していると考えられる。

カーネル起動の 3 つの方法と、ルートファイルシステムの設置場所 3 種類をうまく組み合わせれば、理論的には 9 つの起動方法が実現できる。さらに Toshiba AC100(Dynabook AZ)などの NVIDIA Tegra2 の環境では、ユーザーデータ領域として data 領域よりもさらに大きな storage 領域という部分もあるため、選択肢がより広がる。

なお、通常利用するデスクトップ環境含めた Linux 環境を構築するとすれば最低でも約 2GB 以上の容量が必要になる。そのため、system 領域や userdata 領域上で動かす場合は、ファイルの選択を必要最小限にしたり、cramfs や squashfs などの圧縮ファイルシステムと組み合わせるなどして、ルートファイルシステムを小さくする工夫が必要となる。

4. 移植例: Nexus One

ここでは1つの移植例として Google Nexus One 上で MeeGo 動かす方法を解説する。MeeGo とは、Intel 社を中心に開発されていた Moblin と 、Nokia 社が開発していた Maemo を統合した、次世代コンピュータ機器のためのプラットフォームである。現在、Netbook やデスクトップ PC、スマートフォンを始めとした携帯端末、車載機器やインターネット TV などがターゲットとされており、CPU として x86(SSSE3 の命令セットを持つ Atom, Core 2 以降)と ARM(armv7l, Cortex-A8 以降等)がサポートされている。ARM が動作する Linux ディストリビューションは Fedora や Ubuntu、Debian GNU/Linux などいくつか存在するが、Nexus One 用ということで、ここではスマートフォンに必要な機能、ユーザーインターフェイスがそろった

MeeGo を移植する。

移植の手順を「図 3:MeeGo 移植の流れ」に示す。この手順は、「方法 2: 書き込みツール経由で独自カーネルを転送」と「場所 3: SD カードを利用」とを組み合わせる方法である。PC 側は Ubuntu 10.04 を使って行うことを想定している。

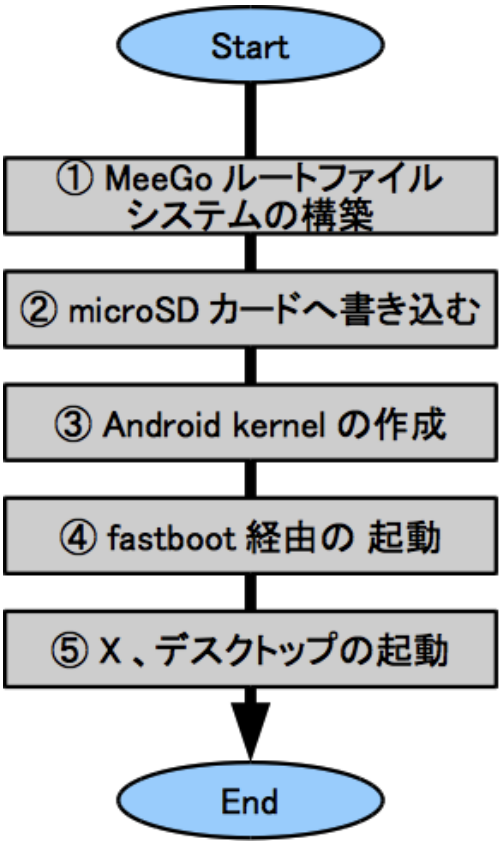

図 3:MeeGo移植の流れ

① MeeGo ルートファイルシステムの構築

まずは MeeGo のルートファイルシステムを作成する。ルートファイルシステムのイメージ作成には MeeGo プロジェクトから提供されている「Image Creator」を使う。まずは PC の/etc/apt/sources.list に、以下の 1 行（Image Creator があるリポジトリの URL）を追加する。

```
deb http://repo.meego.com/tools/repos/ubuntu/10.04/ /
```

続いて Image Creator をインストールする。これは mic2 というパッケージ名で提供されている。

```
$ sudo apt-get update
$ sudo apt-get install mic2
```

次に kickstart ファイルを作成する。kickstart は Fedora などで使われているインストーラ「Anaconda」の自動インストールを行うために作られたもので、パーティション構成や、RPM の入手先、インストールするパッケージなどを設定することができる。今回は Nexus One 用のイメージを作るために「リスト 1：meego-nexusone.ks」のような内容で作成する。

kickstart ファイルのフォーマットは FedoraProject のページ[v]を参考にすることができる。

リスト1：meego-nexusone.ks

```
# 言語の設定
lang en_US.UTF-8

# キーボードレイアウトの設定
keyboard us

# タイムゾーンの設定
timezone --utc America/Los_Angeles

auth --useshadow --enablemd5

# microSDの第1パーティションとして1.6GBの
# ext3フォーマットで/を構築する
part / --size=1600  --ondisk mmcblk0p --fstype=ext3

# microSDの第2パーティションとして256MBの
# swapパーティションを構築する
part swap --size=256 --ondisk mmcblk0p --fstype=swap

# rootのログインパスワードをmeegoとする
rootpw meego

xconfig --startxonboot
desktop --autologinuser=meego  ¥
  --defaultdesktop=DUI ¥
  --session=/usr/bin/mcompositor

user --name meego  --groups audio,video ¥
  --password meego

# MeeGoのコアパッケージの最新を参照する
# Yumリポジトリ URLを参照
repo --name=core
--baseurl=http://repo.meego.com/MeeGo/builds/trunk/preview/core/
repos/armv7l/packages/ ¥
--save --debuginfo --source ¥
--gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-meego

# MeeGoのHandsetパッケージの最新を参照する
# Yumリポジトリ URLを参照。Handsetとは携帯端末向けに
# 作られたコンポーネントを表す
repo --name=handset ¥
--
baseurl=http://repo.meego.com/MeeGo/builds/trunk/preview/handset
/repos/armv7l/packages/ ¥
--save --debuginfo --source ¥
--gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-meego

# インストールするパッケージのリストを以下に記述する
# @から始まるものはパッケージグループであり、
# ひとまとまりにパッケージを指定できる。
%packages
@MeeGo Core
@Minimal MeeGo X Window System
@MeeGo Handset Desktop
@MeeGo Handset Applications

xorg-x11-server-Xorg-setuid

# デバッグ用にopenssh-serverやxterm、strace、
# gdbなどのパッケージを追加する
openssh-server
xterm
wget
qt-demos
connman-test
strace
sensorfw
libmeegotouch-devel
bootchart
xorg-x11-utils-xev
xdg-users-dirs
gdb
yum
yum-utils

# Xを表示するためのビデオドライバとして
# fbdevをを使う
xorg-x11-drv-fbdev

# XのDRIを有効にするためにswrastを使う
mesa-dri-swrast-driver
%end

# パッケージインストール後の処理。chrootされた
# 状態で実行され、シェル形式で記述可能。
%post

# MeeGo Handsetのインターフェイスを表示するための
# 設定。NexusOneというテーマを設定する
Config_Src=`gconftool-2 --get-default-source`
gconftool-2 --direct --config-source $Config_Src ¥
  -s -t string /meegotouch/theme/target NexusOne

# 前述のNexusOneというテーマを定義する。
# 解像度の設定でに合わせた480x800に合わせる。
# これによってMeeGo Handsetのインターフェイスが
# このサイズに調整される
cat >>/etc/meegotouch/devices.conf <<EOF
[NexusOne]
resolutionX=480
resolutionY=800
ppiX=256
ppiY=256
showStatusBar=false
EOF

# X起動時に実行するスクリプトをオリジナルで作成する。
# Hardware Renderingができないため、Software
# Rendering(引数-software)を使って表示するようにする。
cat >/usr/bin/startdui <<EOF
#!/bin/sh
/usr/bin/mthemedaemon &
/usr/bin/sysuid -software -remote-theme &
/usr/bin/meego-im-uiserver -software -remote-theme &
/usr/bin/mdecorator -software -remote-theme &
/usr/bin/duihome --desktop -software -remote-theme &
exec /usr/bin/xterm
EOF
chmod +x /usr/bin/startdui

# gconfのデフォルト設定をNexusOneに置き換える
sed -i 's!aava!NexusOne!g' ¥
/etc/gconf/gconf.xml.defaults/%gconf-tree.xml

# 前述のスクリプトをX起動時に実行するように
# /etc/inittabを編集する
sed -i 's!¥/usr¥/sbin¥/meego-dm!¥/usr¥/bin¥/xinit ¥
¥/usr¥/bin¥/startdui!' /etc/inittab

%end

# パッケージインストール後の設定。
# MeeGoのリリースバージョンを記述する
%post --nochroot
if [ -n "$IMG_NAME" ]; then
    echo "BUILD: $IMG_NAME" ¥
      >> $INSTALL_ROOT/etc/meego-release
fi

%end
```

meego-nexusone.ks を作成後、Image Creator を実行する。コマンドは mic-image-creator で引数に kickstart ファイル、イメージフォーマットに raw フォーマット(dd コマンドで書き込むことでブータブルなディスクとして使える)を追加する。このツールはマルチバイト環境だと実行中にエラーになるため、LANG=C を設定して実行している。

```
$ sudo LANG=C mic-image-creator ¥
-c meego-nexusone.ks -f raw ¥
--run-mode=0 --arch=armv7l
```

実行すると RPM パッケージがダウンロードされ、インストール、イメージの作成が行われる。本稿執筆時点では以下のようなファイルが作成された。

```
meego-nexusone-1.1.80.20101024.2322-mmcblk0p.raw
```

② microSD カードへ書き込む

作成したイメージを dd コマンドで SD カードに書き込む。kickstart パーティションのサイズに合わせるため、2GB 以上の SD カードを用いる。

```
dd if=meego-nexusone-1.1.80.20101024.2322-mmcblk0p.raw ¥
of=/dev/sdx＜microSD のデバイスファイル＞ bs=4k
```

③ Android カーネルの作成

Nexus One 用のカーネルを構築する。カーネルは Android の git ツリー[vi]で公開されているものを使う。

```
$ git clone git://android.git.kernel.org/kernel/msm.git
$ cd msm
```

続いてブランチをチェックアウトする

```
$ git checkout -b nexusone origin/android-msm-2.6.29-nexusone
```

次にカーネル設定を用意する。Nexus One は mahimahi と呼ばれるプラットフォームに該当する。デフォルトのカーネル設定を.config としてコピーする。

```
$ cp arch/arm/configs/mahimahi_defconfig .config
```

そして、.config ファイルを以下のように編集する。

Android は initrd を使うため BLK_DEV_INITRD =y となっているが、今回は fastboot から転送したカーネルをロードするので initrd を使わない。この設定を無効にしておく。

```
-CONFIG_BLK_DEV_INITRD=y
+# CONFIG_BLK_DEV_INITRD is not set
```

デフォルトでは無線LANドライバのファームウェアのパスが Android のパスになっているため一般的なパスに変更する。

```
-CONFIG_BCM4329_FW_PATH="/system/etc/firmware/fw_bcm4329.bin"
+CONFIG_BCM4329_FW_PATH="/lib/firmware/fw_bcm4329.bin"
```

Android はフレームバッファで画面の描画を行うが、MeeGo は X.Org を使うため、VT(Virtual Terminal、仮想端末)を有効にする。

```
-# CONFIG_VT is not set
+CONFIG_VT=y
+CONFIG_CONSOLE_TRANSLATIONS=y
+CONFIG_VT_CONSOLE=y
+CONFIG_VT_HW_CONSOLE_BINDING=y
```

逆にフレームバッファを使わなくなるので不要な設定を無効にする。

```
+# CONFIG_VGA_CONSOLE is not set
+# CONFIG_FRAMEBUFFER_CONSOLE is not set
```

バイナリはクロスコンパイルで作成する。クロスコンパイラは Android の prebuild ツリー[vii]にあるプログラムを使用する。まずは実行パスを通しておく(以下はコマンドライン 1 行)。

```
$ export PATH=$PATH:＜Android のソースツリー＞
/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin
```

次にコンパイルを行う。カーネルイメージは zImage フォーマットで作成する。

```
$ make oldconfig ARCH=arm CROSS_COMPILE=arm-eabi-
$ make zImage modules ARCH=arm CROSS_COMPILE=arm-eabi-
```

モジュールもコンパイルされ、無線LANドライバである bcm4329.ko が作られる。

```
./drivers/net/wireless/bcm4329/bcm4329.ko
```

④ fastboot 経由の起動

まず fastboot を用意する。Android をビルドすると出来上がるのでここからコピーする。

```
$ cp <Android のソースツリー>/out/host/linux-x86/bin/fastboot .
```

続いて fastboot コマンドを使って Nexus One を認識出来るように udev のルールを追加する。Nexus One は fastboot が使える状態で PC に接続すると Vendor ID が 0bb4 として認識される。

以下は、一般ユーザによってアクセス出来るようにパーミッションを 0666 に変更するという設定である。

```
$ sudo vi /etc/udev/rules.d/99-android.rules
SUBSYSTEM==”usb”, SYSFS{idVendor}==”0bb4”, MODE=”0666”
```

そして udev を再起動する。

```
$ sudo restart udev
```

次に Nexus One を USB ケーブルで接続する。その後、電源ボタンを長押しして電源 OFF にする。電源が落ちたことを確認したら電源ボタン+ボリュームダウンボタンを押す。すると HBOOT というメニューが表示される。ボリュームキーを使うとカーソルを移動できるので、FASTBOOT を電源ボタンで選択する。これで PC から fastboot コマンドが実行できる状態になった。

fastboot コマンドを使って以下のようにデバイス情報が見つかれば利用可能である。

```
$ ./fastboot devices
????????????    fastboot
```

接続が確認できたら fastboot コマンドを使って zImage を転送し、さらにカーネルパラメータを渡して起動する。

なお、ルートファイルシステムは SD カードの第 1 パーティションなので、root=/dev/mmcblk0p1 と指定し、ファイルシステムのフォーマットは ext3 なので rootfstype=ext3 と指定することに注意。

```
$ ./fastboot ¥
-c ’init=/sbin/init rootwait root=/dev/mmcblk0p1 ¥
rootfstype=ext3 rw’ boot zImage
creating boot image...
creating boot image - 2240512 bytes
downloading ’boot.img’... OKAY
booting... OKAY
```

転送が完了すれば MeeGo の起動が始まる。

⑤ X、デスクトップの起動

起動メッセージは出力されないが 1 分近く待つと X が起動し、MeeGo Handset のインターフェイスが表示される（写真1）。

また、タッチパネル用のドライバ「tslib」が自動的に検出されて使用できるようになる。今回のテスト機では、補正は特にする必要がなかった。無線 LAN は、先ほど作成した bcm4329.ko というモジュールをロードすれば利用可能になる。

MeeGo Handset は、元々 EGL(Embedded-System Graphics Library)でハードウェアレンダリングを利用することが想定されている。しかし Nexus One ではハードウェアレンダリングのための情報が少なく、これを使えるようにするためには Android のライ

ブラリを解析が必要となる。現状ではソフトウェアレンダリングによって表示させているが、非常に動作が重く、いわゆる「コマ送り」的な動きとなってしまっている。

5. 今後の課題

Nexus One では MeeGo の起動と、画面の表示まで確認することができた。この先は周辺機器を使えるようにする作業が必要になる。現時点で課題となるのは以下の点である。

①ハードウェアレンダリングを有効にする

グラフィックアクセラレーターの部分はどのメーカーもソースコードを公開していないため、場合によってはクローズドなライブラリやドライバ、ファームウェアをオリジナルの Android から抽出して置き換えるなどの作業が必要になるだろう。

②音声入出力やカメラ等の周辺装置への対応

このままでは、音声の入出力が行えず、またカメラなども動作しない。ドライバだけでなく、これらを利用するためのアプリケーションを用意する必要がある。

③電話機能を使えるようにする

SIM カード使って通話や SMS を送受信する場合、発呼を行うための AT コマンドを調べる必要がある。製造メーカーが公開していれば参考にできるが、公開していない場合は容易に利用可能にならない。

④ハードウェアボタンに特定の機能を割り当てる

多くの Android 端末は、「ロック」、「ボリュームアップ/ダウン」、「戻る」、「メニュー」、「ホーム」、「検索」、「通話」等のボタンスイッチを装備しており、それらを特定の機能に割り当てることができる。これらは Android で便利なように用意されているものだが、MeeGo の場合は、これらすべてを割り当てる必要はないかもしれない。

⑤フラッシュメモリから MeeGo を起動する

即ち、「方法 1: boot 領域を上書きし、通常起動する」と「場所 1: system 領域を上書き」を組み合わせて起動させるというものである。ある程度実用レベルで動作するようになれば最終的に Android と入れ替えれば、完全な移植と言えるだろう。

本稿では Nexus One への移植を解説したが、他のプラットフォームの場合、どのような課題があるだろうか。

●Dynabook AZ(Toshiba AC100)の場合:

Dynabook AZ は、システムチップセットとして NVIDIA Tegra2 を採用している。Tegra については、NVIDIA 社から Linux For Tegra(L4T)という開発ツールキットが提供されており、これを活用することができる。また、メーカーである東芝からは、Android ソースコード（GPL の部分のみ）が提供されており、カーネルもその中に含まれているため、単純に起動するだけであれば作業は比較的容易である。しかし、MeeGo として提供されている armv7l バイナリが、Tegra2 でサポートされていない命令セットを呼び出しているため、そのままでは動作しないソフトウェアがある。これらは Android のクロスコンパイラを使って Tegra2 用に再コンパイルしたものを用意する必要がある。ARM プラットフォームの場合、x86 よりバイナリ互換性が低いので、すべてのプログラムがそのまま正しく動作するとは限らない。必要に応じてソースコードからコンパイルするケースが出てくるだろう。

6. まとめ

本論文では Android 端末の構造と移植の方法をまとめた。Android 端末だけでなく、組み込み機器のほとんどは基本的にクローズドな部分が存在しており、すべてオープンソースソフトウェアだけで動かすには現実的に難しい点が出てくるだろう。その上、Android の場合、オープンなプラットフォームと謳いながらもグラフィックス機能など重要な点はやはりクローズドであるため、今までの組み込み Linux と変わらないともとれる。しかし、Android を使って各メーカーが製品を作り上げたことで一定の共通化はできるようになった。クローズドな機能があっても一般的な Linux で再利用出来るなら Android 端末の新たな可能性を見つけられるかもしれない。特に新しい OS がサポートされない古い端末であれば、移植による「延命」は新しい価値につながる可能性がある。

i http://www.linux-mtd.infradead.org/
ii git://android.git.kernel.org/platform/system/core.git
iii git://android.git.kernel.org/platform/system/core.git
iv git://android.git.kernel.org/platform/external/yaffs2.git
v http://fedoraproject.org/wiki/Anaconda/Kickstart
vi git://android.git.kernel.org/kernel/msm.git
vii git://android.git.kernel.org/platform/prebuilt.git