Tutorial su Openmoko

Giorgio Ravera

A.A 2007/2008

Indice

Introduzione

L'obiettivo di questo documento è guidare l'utente nell'utilizzo dell'ambiente di sviluppo fornito a supporto della piattaforma Openmoko (`http://www.openmoko.org/`).

Tale studio è stato svolto nell'ambito del corso di Architetture e Protocolli per Reti Wireless, tenuto dal Prof. Raffaele Bolla in collaborazione con Ing. Matteo Repetto.

Capitolo 1

Dati Generali

1.1 Openmoko

OpenMoko è un progetto per la creazione di una piattaforma aperta per smartphone, usando software libero. L'obiettivo è quello di creare una general-purpose distribuzione GNU/Linux per telefoni cellulari.

OpenMoko utilizza il kernel Linux, con un ambiente grafico sviluppato utilizzando X.Org, GTK+ toolkit e Matchbox window manager. Il framework OpenEmbedded e il sistema di gestione dei pacchetti opkg (derivata da ipkg) sono usati per creare e mantenere i pacchetti software.

Il progetto è stato annunciato nel 2006 dai suoi fondatori First International Computer (FIC), un'azienda di elettronica con sede in Taiwan.

1.2 Neo1973 e Neo FreeRunner

1.2.1 GTA01

Il Neo1973 è il primo telefono progettato per utilizzare OpenMoko. E' prodotto dalla FIC. Fece il suo ingresso nel mercato il 9 luglio 2007.

Figura 1.1: Neo1973

Viene distribuito in due modalità:

- **Neo Base**: tutto quello di cui hanno bisogno gli sviluppatori di applicazioni mobili per divertirsi:
  - Neo 1973 (GTA01B v4)
  - Batteria
  - Stiletto
  - Headset
  - Caricatore AC
  - Sacchetto
  - Cordino
  - Scheda SanDisk 512MB MicroSD
  - Cavo di connettività Micro USB
- **Neo Advanced**: tutto quello che desidera un hacker di dispositivi mobili per entrare a fondo nel primo telefono free:
  - Neo1973 (GTA01B v4)
  - Batteria (2x)
  - Stiletto
  - Headset
  - Caricatore AC
  - Sacchetto
  - Cordino
  - Scheda SanDisk 512MB MicroSD (2x)
  - Cavo di connettività Mini USB (2x)
  - Cavo USB Host Mode
  - Cavo Debug Flex
  - Debug Board v2 (JTAG e console seriale)
  - Ruggedized Toolbox con attaccatura da spalla
  - Guitar Pick (per aprire il case)
  - Cacciavite Torx T6

  Non è necessario il pacchetto Avanzato per usare il kernel normale, o sviluppare applicazioni.

1.2.2 GTA02

GTA02 (Il Mass Market Neo 1973) è ancora in fase di test ed è prevista la sua uscita nell'ottobre 2008. Avrà i seguenti nuovi componenti hardware:

- 802.11 b/g WiFi
- Samsung 2442 SoC
- Acceleratore grafico SMedia 3362
- 2 Accelerometeri 3D
- Memoria flash da 256MB
- Batteria da 1700mAh
- Una CPU più veloce (S3C2442/400)

1.3 Framework

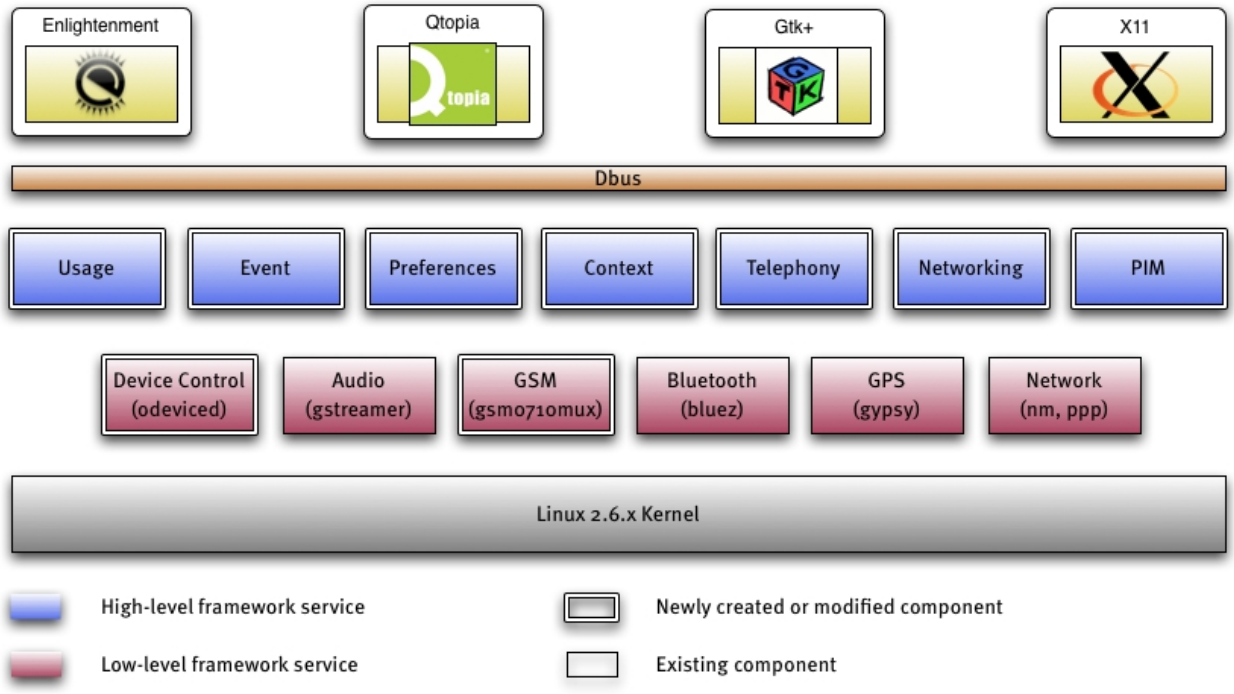

Figura 1.2: Framework

1.4 Ambiente di Cross Compilazione

Per poter compilare software sull'Openmoko è necessario creare un ambiente di cross compilazione. Si possono scegliere due strade:

1. creare un ambiente manualmente tramite crossdev (capitolo 2)
2. utilizzare MokoMakefile (capitolo 3)

Capitolo 2

Ambiente di Cross Compilazione

2.1 Introduzione

Per completezza di analisi segue una descrizione di come realizzare un ambiente di cross-scompilazione.

Con il termine cross compilazione si intende la tecnica mediante la quale si compila un codice sorgente e si ottiene un file eseguibile binario con l'ausilio di un elaboratore con architettura diversa da quella della macchina finale sullla quale verrà utilizzato il binario.

Al fine di realizzare tale ambiente è necessario compilare una differente versione dei seguenti software:

- gcc - compilatore GNU per il linguaggio C
- binutils - utility per linking delle librerie
- libc - librerie per il linguaggio C (glibc o ulibc)
- linux-headers - header del kernel richieste dalle libc
- gdb - software per debugging dei binari

A differenza degli analoghi software già presenti nel sistema, questi saranno eseguibili dalla coppia sistema operativo e processore installati ma produrranno codice binario per l'architettura target.

Si può realizzare un ambiente di cross compilazione attraverso l'uso di crossdev (`http://crossdev.org/`), un tool opportunamento realizzato per questo scopo.

Gli step da effettuare sono i seguenti:

1. installazione crossdev
2. compilazione della toolschain
3. creazione dell'ambiente

2.2 Installazione di Crossdev

Gentoo usa un software di gestione software chiamato emerge. Si configura attraverso il file **/etc/make.conf**. Tale software sfrutta il concetto di installatore di pacchetti di rete come apt, ma si differisce da quest'ultimo per il fatto che i software sono disponibili in sorgenti e vengono opportunamente compilati.
Per installare crossdev è necessario eseguire il seguente comando da utente root:

```
# emerge crossdev
```

Una volta fatto ciò bisogna identificare l'architettura target per la quale crosscompilare del software.

2.3 Compilazione della Toolchain

Gentoo usa una serie di variabili, presenti in **/etc/make.conf**, volte a dare istruzioni a gcc per compilare il sistema:

- CHOST: sistema target (es: x86_64-pc-linux-gnu)
- CFLAGS: indica a gcc come compilare il software (es: -march=nocona -O2 -pipe)
- USE: indica a gcc che moduli del software abilitare e quali non abilitare (es: gnome network -kde -mysql - ldap)
- ROOT: indica la directory in cui andare a installare il software (es: /)

Per prima cosa è necessario capire di che valore per CHOST si ha bisogno. La sintassi di tale variabile è la seguente:

```
<cpu_type>-<something>-<operating system>-<libc>
```

Nel nostro caso si è scelto di compilare per un sistema arm a 32 bit con linux e gnueabi come librerire C. Pertanto la stringa sarà la seguente:

```
armv4t-angstrom-linux-gnueabi
```

Da notare che non si è usato il secondo argomento (¡something¿): di solito è un valore usato per dare un nome al sistema senza e può essere omesso.
Ora è possibile procedere con l'installazione dell'ambiente usando crossdev nel seguente modo:

```
# USE="-*" crossdev -p -v \
--binutils <versione> \
--gcc <versione> \
--kernel <versione> \
--libc <versione> \
--ex-gdb \
--target <target>
```

Analizziamo l'uso di **USE=-*** aiuta a risolvere eventuali problemi di mascheramento di pacchetti per specifiche architetture. In Gentoo un pacchetto in fase di test per il quale non si ha la certezza della stabilità viene indicato come mascherato e può esserlo solo per una particolare architettura.
Per ogni software che si installa con quel comando (gcc, libc, kernel e binutils) è possibile richiederne una particolare versione. Se non venisse specificato nulla di default viene scelta l'ultima release disponibile.

Spesso tale comando può portare ad esiti spiacevoli a causa di incompatibilità tra le versioni richieste o errori nella scelta di target o versioni. L'opzione **-p** non installa il software ma si limita ad effettuare un test di coerenza su ciò che si vuole installare seguito dalla presentazione di un report.
L'output potrebbe essere simile al seguente:

```
portatile usr # USE="-*" crossdev \
 --ex-gdb \
 --target armv4t-angstrom-linux-gnueabi
--------------------------------------------------------------------------------
 * Host Portage ARCH:     amd64
 * Target Portage ARCH:   arm
 * Target System:         armv4t-angstrom-linux-gnueabi
 * Stage:                 4 (C/C++ compiler)

 * binutils:              binutils-[latest]
 * gcc:                   gcc-[latest]
 * headers:               linux-headers-[latest]
 * libc:                  glibc-[latest]
 * Extra: gdb:            DO IT

 * PORTDIR_OVERLAY:       /usr/local/layman/connectical
 * PORT_LOGDIR:           /var/log/portage
 * PKGDIR:                /usr/portage/packages/cross/armv4t-angstrom-linux-gnueabi
 * PORTAGE_TMPDIR:        /var/tmp/cross/armv4t-angstrom-linux-gnueabi
  _  _  ~  _  _  _  ~  _  _  _  ~  _  _  _  ~  _  _  _  ~  _  _  _  ~  _  _  _
 * Forcing the latest versions of {binutils,gcc}-config/gnuconfig ...     [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-binutils.log
 * Emerging cross-binutils ...                                           [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-gcc-stage1.log
 * Emerging cross-gcc-stage1 ...                                         [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-linux-headers.log
 * Emerging cross-linux-headers ...                                      [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-glibc.log
 * Emerging cross-glibc ...                                              [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-gcc-stage2.log
```

```
 * Emerging cross-gcc-stage2 ...                                          [ ok ]
 * Log: /var/log/portage/cross-armv4t-angstrom-linux-gnueabi-gdb.log
 * Emerging cross-gdb ...                                                 [ ok ]
```

Se tutto va a buon fine e si vuole procedere all'installazione basta rimuovere l'opzione -p dal comando precedente e attendere il completamento della compilazione.

A compilazione terminata avremo una directory /usr/armv4t-angstrom-linux-gnueabi che costituirà la SYSROOT per qualsiasi software che si vorrà cross compilare.
A questo punto si è pronti per cross compilare semplici software da riga di comando richiamanto il nuovo compilatore appena compilato nel seguente modo:

```
#  powerpc-ppc-linux-gnu-<compiler> <source> <flags>
```

ove <compiler> può essere gcc, g++, gfortran e altri e <source> il codice sorgente mentre con <flags> si indicano le opzioni di compilazione analogamente all'uso per compilazioni standard.

2.4 Creazione dell'Ambiente

Se si volesse cross compilare librerie o pacchetti software disponibili nel portage di gentoo tramite emerge è necessario creare un vero e proprio ambiente che usi la nuova SYSROOT creata e una specifica versione di make.conf.

Per prima cosa è necessario creare una variabile d'ambiente che indichi la SYSROOT. Per fare ciò si può aggiungere una riga al file **/etc/environment** come segue:

```
# echo SYSROOT=/usr/armv4t-angstrom-linux-gnueabi/ >> /etc/environment
```

Fatto ciò si devono creare alcune directory e link simbolici non generate di default all'interno della nuova SYSROOT:

```
# mkdir $SYSROOT/etc
# ln -s /usr/portage/profiles/<SOMETHING> $SYSROOT/etc/make.profile
```

L'ultima riga rappresenta la scelta del profilo da utilizzare per la compilazione. Gentoo dispone di differenti profili che forniscono una versione specifica di variabili d'ambiente (USE in particolare) pensate appositamente per soddisfare richieste eterogenee.
La scelta dipende fortemente da ciò che si intende compilare e dalla piattaforma di riferimento.

Scelto il profilo si passa a realizzare una versione personalizzata make.conf che emerge userà per compilare i software in questo nuovo ambiente. Una versione di esempio potrebbe essere la seguente:

```
ACCEPT_KEYWORDS="arm"
ARCH="arm"
CFLAGS="-O1 -pipe"
CHOST="armv4t-angstrom-linux-gnueabi"
CXXFLAGS="${CFLAGS}"
INPUT_DEVICES="keyboard"
MAKEOPTS="-j2"
USE="arm symlink"
```

L'ultima cosa che resta da fare è realizzare una versione modificata di emerge che compili nel nuovo ambiente. Si può creare il file **/usr/lib/portage/bin/emerge_arm** che contiene le seguenti righe:

```
#!/bin/bash
CBUILD=$(portageq envvar CHOST)
PORTAGE_CONFIGROOT="$SYSROOT"
if [[ "$1" == "--root" ]] ; then
    ROOT="$2"
    shift 2
else
    ROOT="$SYSROOT"
fi
export CBUILD PORTAGE_CONFIGROOT ROOT
emerge \$*
```

Fatto ciò si è in grado di installare componenti software aggiuntivi con il comando:

```
# /usr/lib/portage/bin/emerge\_arm <PACKAGES>
```

Capitolo 3

MokoMakefile

MokoMakefile è un metodo completamente automatizzato per la configurazione dell'ambiente di sviluppo OpenMoko, uno strumento inestimabile che aiuta i nuovi sviluppatori ad ottenere un ambiente con le stesse configurazioni di quelli già presenti. Permette di ottenere la stessa ripetibilità nella compilazione e manutenzione dell'ambiente che OpenEmbedded introduce nell'attuale software embedded.

Il MokoMakeFIle è capace di costruire le immagini sia di OM-2007.1 sia di OM-2007.2 attraverso la modifica di una variabile all'interno del Makefile stesso.
Le principali operazioni che consente di eseguire in totale autonomia sono:

- compilazione di un immagine di root:
  - openmoko-devel-image
  - openmoko-qtopia-image
- aggiornamento delle immagini di root compilate
- installazione sul neo1973 di:
  - u-boot
  - kernel
  - immagine di root
- emulazione del neo1973 tramite QEMU

3.1 Requisiti Hardware

Indipendentemente da qualsiasi MokoMakeFile o da qualsiasi processo manuale per mettere a punto l'ambiente OpenMoko, ci sono molti prerequisiti che dovrebbero essere soddisfatti prima di cominciare la costruzione:

- RAM: la macchina che ospiterà la costruzione (build host) dovrebbe avere almeno 512 Mb di RAM, ed uno stesso ammontare di swap. Alcuni pacchetti costruiti da OpenMoko, come busybox sono costruiti compilando tutti i files sorgente in un solo file binario, e questo causa la crescita del processo di gcc fino a 300 Mb, e nessuna parte della memoria occupata dovrebbe essere in swap perchè il tutto finisca in tempi ragionevoli. Per quanto riguarda busybox, può essere disattivata, ma ciò significherebbe che busybox non sarà ottimizzata per gcc.
- Spazio su disco: occorrono almeno 12 Gb di spazio disponibile su disco perchè la costruzione di OpenMoko vada a buon fine (vedere sotto un consiglio per ridurre la quantità di spazio richiesto).
- Tempo: la costruzione iniziale richiede almeno 5 ore (su un Core2Duo 2 GHz senza ottimizzazione per il multiprocessore) ed può richiedere alcuni giorni per le macchine lente.

3.2 Requisiti Software

Per far funzionare correttamente l'ambiende di compilazione sono necessari alcuni software:

- **subversion**: software per il controllo versione
- **quilt**: software per la gestione delle patch
- **monotone**: software per il controllo versione utilizzato da OpenEmbedded. Monotone non è disponibile in tutte le fistribuzioni Linux. E' possibile scaricarlo dal sito `http://monotone.ca`
- **diffstat**: software usato da OpenEmbedded per tenere traccia dell'output del comando diff
- **texi2html**: software per convertire file Texinfo in html
- **git**: software per il controllo versione
- **psyco**: Python just-in-time compiler. Opzionale ma fortemente raccomandato

3.2.1 Installazione su Debian/Ubuntu e derivati

Per soddisfare i requisiti in ambiente Debian/Ubuntu è necessario eseguire i seguenti comandi da root:

```
# apt-get install subversion build-essential help2man diffstat texi2html texinfo \
  cvs gawk cogito libncurses5-dev zlib1g-dev libssl-dev libgtk2.0-dev \
  ca-certificates python-pysqlite2 sqlite3 sqlite3-doc python-pysqlite2-dbg \
  quilt python-psyco ccache gcc-3.4 g++-3.4 libsdl1.2-dev lynx netpbm dosfstools
```

3.2.2 Installazione su RedHat/Fedora e derivati

Per soddisfare i requisiti in ambiente RedHat/Fedora è necessario eseguire i seguenti comandi da root:

```
# yum install subversion build-essential help2man diffstat texi2html texinfo \
  cvs gawk cogito libncurses5-dev zlib1g-dev libssl-dev libgtk2.0-dev \
  ca-certificates python-pysqlite2 sqlite3 sqlite3-doc python-pysqlite2-dbg \
  quilt python-psyco ccache gcc-3.4 g++-3.4 libsdl1.2-dev lynx netpbm dosfstools
```

3.2.3 Installazione su SuSE e derivati

Per soddisfare i requisiti in ambiente SuSE è necessario eseguire i seguenti comandi da root:

```
# yast -i gcc-c++ ncurses-devel zlib-devel libopenssl-devel gtk2-devel subversion\
  diffstat texinfo help2man
# wget download.opensuse.org/repositories/devel:/tools:/gcc/openSUSE_Factory/i586/\
  {cpp,gcc}33-3.3.3-41.8.i586.rpm
# rpm -Uhv {cpp,gcc}33-3.3.3-41.8.i586.rpm
```

3.2.4 Installazione su Gentoo e derivati

Per soddisfare i requisiti in ambiente Gentoo è necessario eseguire i seguenti comandi da root:

```
# echo 'dev-util/monotone ~'`readlink /etc/make.profile | awk -F / '{print $6}'`\
  >>/etc/portage/package.keywords
# emerge -u subversion quilt monotone diffstat texi2html dev-util/git psyco
```

In Gentoo si suppone che sia già presente un compilatore e le librerie necessarie per il corretto funzionamento del sistema, senza le quali non sarebbe possibile ottenere un installazione della distribuzione stessa.

3.3 Preparazione dell'ambiente

Per prima cosa è necessario creare una directory di lavoro e scaricare una versione aggiornata del MokoMakefile:

```
$ mkdir ~/moko
$ cd ~/moko
$ wget http://www.rwhitby.net/files/openmoko/Makefile
```

Fatto ciò è necessario preparare l'ambiente attraverso l'esecuzione del seguente comando:

```
make setup
```

Tale comando si occupa di scaricare e installare bitbake, software utilizzato da OpenEmbedded per la compilazione dei pacchetti e la realizzazione dei file autoinstallanti ikp, sincronizzarsi con il repository OpenEmbedded e Openmoko, generare i file di configurazione per la compilazione e configurare delle variabili d'ambiende. E' possibile personalizzare alcuni elementi modificando a mano il file:

```
$OMDIR/build/conf/local.conf
```

variando alcuni parametri o aggiungendone di altri. La configurazione di default è la seguente:

```
MACHINE = "om-gta01"
DISTRO = "openmoko"
BUILD_ARCH = "i686"
INHERIT += "rm_work"
```

Per compilare l'immagine su om-gta02 basta variare la variabile MACHINE.
Altre opzioni potrebbero essere le seguenti:

```
# Generazione di locales differenti
ENABLE_BINARY_LOCALE_GENERATION = "1"
GLIBC_GENERATE_LOCALES = "en_US.UTF-8 it_IT.UTF-8"
# Compilazione su più core/processori
PARALLEL_MAKE = "-j 3"
BB_NUMBER_THREADS = "2"
```

Al termine della procedura, avremo nella directory $OMDIR i seguenti file:

```
xraver@bestia:~/moko$ ls
bitbake  build  Makefile  openembedded  openmoko  setup-env  stamps
```

3.4 Compilazione dell'immagine

Una volta configurato l'ambiente si può iniziare la compilazione con il comando:

```
$ make openmoko-devel-image
oppure
$ make openmoko-qtopia-image
```

La compilazione richiede molto tempo. A partire da zero è necessario compilare circa 6000 pacchetti. Un esempio di output potrebbe essere il seguente:

```
xraver@bestia:~/moko$ make openmoko-devel-image
( cd build && . ../setup-env && \
  ( bitbake openmoko-devel-image u-boot-openmoko ) )
NOTE: Removed the following variables from the environment:GNOME_DESKTOP_SESSION_ID,
 LESSOPEN,WINDOWPATH,MAKEFLAGS,SSH_AGENT_PID,SHLVL,WINDOWID,OMDIR,MFLAGS,
 GDM_XSERVER_LOCATION,PYTHONPATH,GDMSESSION,LESSCLOSE,OLDPWD,GDM_LANG,
 HISTCONTROL,MAKELEVEL,LS_COLORS
NOTE: Handling BitBake files: | (5616/5616) [100 %]
NOTE: Parsing finished. 5371 cached, 0 parsed, 245 skipped, 0 masked.
NOTE: Cache is clean, not saving.
NOTE: build 200807011457: started

OE Build Configuration:
BB_VERSION        = "1.8.11"
METADATA_BRANCH   = "<unknown>"
METADATA_REVISION = "<unknown>"
TARGET_ARCH       = "arm"
TARGET_OS         = "linux-gnueabi"
MACHINE           = "om-gta01"
DISTRO            = "openmoko"
DISTRO_VERSION    = "P1-Snapshot-20080701"
TARGET_FPU        = "soft"

NOTE: Resolving any missing task queue dependencies
NOTE: preferred version 2.6.1 of glibc not available
 (for item virtual/arm-angstrom-linux-gnueabi-libc-for-gcc)
NOTE: preferred version 115 of udev not available (for item udev-utils)
NOTE: preferred version 115 of udev not available (for item udev)
NOTE: preferred version 115 of udev not available (for item udev)
NOTE: multiple providers are available for virtual/libqte2
 (qte-mt-static, qte, qte-mt);
NOTE: consider defining PREFERRED_PROVIDER_virtual/libqte2
NOTE: Preparing runqueue
NOTE: Executing runqueue
...
NOTE: Running task 95 of 6139 (ID: 1054, /home/xraver/moko/openembedded/
 packages/libtool/libtool-native_1.5.10.bb, do_configure)
NOTE: package libtool-native-1.5.10: started
NOTE: package libtool-native-1.5.10-r10: task do_configure: started
NOTE: package libtool-native-1.5.10-r10: task do_configure: completed
NOTE: package libtool-native-1.5.10: completed
...
```

Una volta terminata la compilazione l'immagine sarà disponibile nella directory:

```
$OMDIR/build/tmp/deploy/glibc/images/om-gta01
oppure
$OMDIR/build/tmp/deploy/glibc/images/om-gta02
```

a seconda che si sia scelto om-gta01 o om-gta02. Il nome dell'immagine sarà il seguente:

```
Openmoko-openmoko-devel-image-glibc-ipk-P1-Snapshot-YYYYMMDD-om-gta0X.rootfs.jffs2
```

ove:

- YYYY: anno
- MM: mese
- DD: giorno
- X: 1 o 2

La data è da intendersi come data di fine di compilazione. Insieme all'immagine viene compilato anche l'u-boot presente anch'esso nella medesima directory.

Se si volesse andare a prelevare solamente i file .ipk si possono trovare in questa directory:

```
$OMDIR/build/tmp/deploy/glibc/ipk
```

tale directory di fatto rappresenta un repository accessibile remotamente dallo stesso telefono per l'upgrade automatico. Per rendere possibile ciò è necessario allestire un webserver o ftpserver sul sistema e far puntare la directory sopra citata.

E' possibile anche prelevare informazioni sull'immagine appena creata andando nella directory:

```
$OMDIR/build/tmp/deploy/glibc/images/om-gta01/ \
 Openmoko-openmoko-devel-image-glibc-ipk-P1-Snapshot-YYYYMMDD-om-gta0X-testlab
```

e leggere i file di testo contenuti in essa che provvederanno a fornire dati quali i pacchetti presenti, le dipendenze, i file presenti e molti altri ancora.

E' infine possibile accedere ai file presenti nell'immagine in due modo:

- estraendo il contenuto dell'immagine stessa (lento)
- esplorando la directory $OMDIR/build/tmp/rootfs

3.5 Compilazione del Kernel

Nella compilazione della rootfs è compresa la compilazione del pacchetto **linux-openmoko**, ovvero il kernel vero e proprio. La sua compilazione comporta la creazione della uImage, ovvero un immagine contenente il file vmlinuz da poter installare nella partizione del neo1973 ad esso dedicata. Essendo un pacchetto di bitbake, oltre all'immagine (che sarà disponibile nella directory $OMDIR/build/tmp/deploy/glibc/images/om-gta0X) verranno installati alcuni file nella rootfs. Quindi è importante mantenere coerenza tra le due immagini così compilate.

Per poter visionare la configurazione del kernel è necessario visionare il file:

```
$OMDIR/build/tmp/work/om-gta01-angstrom-linux-gnueabi/ \
 linux-openmoko-<VERSION>/defconfig
```

Se si volesse personalizzare la configurazione del kernel è necessario modificare a mano il file utilizato da bitbake per la compilazione:

```
$OMDIR/openembedded/packages/linux/linux-openmoko-<VERSIONE>.bb
```

Si tenga presente che la working directory di compilazionem del kernel è

```
$OMDIR/build/tmp/work/om-gta01-angstrom-linux-gnueabi/linux-openmoko-<VERSION>/git
```

3.6 Aggiornamento dell'immagine

E' possibile utilizzare una procedura automatizzata, fornita sempre dal Makefile per aggiornare l'immagine senza dover ricompilare il tutto. Per fare ciò è possibile eseguire in sequenza i seguenti comandi:

```
$ make update-makefile && make update setup openmoko-devel-image
oppure
$ make update-makefile && make update setup openmoko-qtopia-image
```

Il primo comando si occupa di scaricare una eventuale nuova versione del Makefile, il secondo aggiorna i repository e inizia la compilazione dell'immagine a partire dalle nuove versioni dei software necessari.

3.7 Flash del Neo1973

Una volta realizzata l'immagine è possibile inserirla nel telefono. Si può fare questa operazione utilizzando dfu-util, un software fornito dall'ambiente ma anche disponibile in rete.

Sono necessari 3 elementi da installare nel telefono:

- u-boot: il bootloader

- uImage: il kernel
- rootfs: l'immagine di root appena compilata.

Per caricare automaticamente le immagini si possono usare i seguenti comandi:

```
$ make download-images
$ make flash-neo-official
oppure
$ make flash-neo-local
```

I primi due comandi provvederanno a scaricare dal web le immagini più recenti e a caricarle direttamente sul telefono.
E' importante che quest'ultimo sia connesso alla porta usb del computer dal quale si effettua il flash e sia in modalità debug, ovvero che sia stato acceso tramite la pressione contemporanea dei due pulsanti che si trovano ai lati (uno nero, quello di accensione, e uno rettangolare sull'altro lato).
Il secondo si occuperà invece di andare nella directory:

```
$OMDIR/build/tmp/deploy/glibc/images/neo1973
```

e caricare l'immagine più recente. Prima di procedere è necessario eliminare la precedente versione di tale directory e crearne un link simbolico a om-gta01 che conterrà, appunto, l'immagine appena compilata:

```
xraver@bestia:~/moko$ cd $OMDIR/build//tmp/deploy/glibc/images/
xraver@bestia:~/moko/build/tmp/deploy/glibc/images$ ls
i686  neo1973  om-gta01
xraver@bestia:~/moko/build/tmp/deploy/glibc/images$ rm -fr neo1973
xraver@bestia:~/moko/build/tmp/deploy/glibc/images$ ln -s om-gta01 neo1973
```

E' importante osservare che il telefono va acceso in modalità debug, ovvero premendo entrambi i pulsanti presenti (uno sul lato destro, uno sul lato sinistro)

La procedura automatica non aggiorna l'u-boot. Per chi volesse effettuare il flash manualmente i comandi sono i seguenti:

- dfu-util -a u-boot -R -D u-boot.bin
- dfu-util -a kernel -R -D uImage.bin
- dfu-util -a rootfs -R -D rootfs.jffs2

Si noti che dfu-util è presente nella directory $OMDIR/build/tmp/deploy/i686/.
Ecco un esempio di flash dell'u-boot:

```
xraver@bestia:~/moko/build/tmp/deploy/glibc/images/neo1973$ sudo ../i686/dfu-util \
-a u-boot -R -D `ls u-boot-*.bin | head -1`
[sudo] password for xraver:
dfu-util - (C) 2007 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Opening USB Device 0x0000:0x0000...
Claiming USB DFU Runtime Interface...
Determining device status: state = appIDLE, status = 0
Device really in Runtime Mode, send DFU detach request...
Resetting USB...
Opening USB Device...
Found Runtime: [0x1457:0x5119] devnum=12, cfg=0, intf=0, alt=1, name="u-boot"
Claiming USB DFU Interface...
Setting Alternate Setting ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
Transfer Size = 0x1000
bytes_per_hash=4377
Starting download: [##################################################] finished!
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
Resetting USB to switch back to runtime mode
```

E' importante notare che tali comandi devono essere eseguiti con priviliegi di root. Il MokoMafile richiama tali comandi passando per sudo. Per questo motivo è essenziale configurare l'utente locale per poter usufruire correttamente di tali privilegi.

3.8 Emulazione con QEMU

MokoMakefile automatizza la compilazione, il flashing e l'esecuzione di QEMU opportunamente configurato per emulare perfettamente il Neo1973. E' possibile, anche in questo caso, utilizzare sia le immagini scaricate automaticamente da internet oppure utilizzare la propria, appena compilata. Per un corretto funzionamento è necessario avere installati i seguenti pacchetti:

- lynx
- netpbm
- sdl-devel
- gcc versione 3

Per installarli in Debian/Ubuntu:

```
# apt-get install lynx netpbm libsdl1.2-dev gcc-3.*
```

Per installarli in Gentoo

```
# emerge lynx netpbm libsdl
# cd /usr/portage/sys-devel/gcc
# emerge ‘ls |grep gcc-3. |tail -1‘
```

Una volta installati i pacchetti necessari è possibile iniziare con la compilazione di qemu. Per fare questo si può procedere come segue:

```
$ make qemu
oppure
$ make qemu-local
```

Entrambi i comandi provvederanno a compilare QEMU, flashare un immagine del kernel e della rootfs e lanciarne l'esecuzione. La differenza sta nel fatto che il primo comando userà le ultime versioni disponibili su internet, mentre il secondo utilizzerà quelle compilate localmente, presenti in $OMDIR/build/tmp/deploy/glibc/images/neo1973.

E' indispensabile la presenza di un compilatore gcc versione 3. Non è necessario averlo utilizzato per compilare la rootfs o il kernel stesso neppure che sia quello di default del sistema. Deve però essere presente e localizzabile al momento della compilazione dei sorgenti di QEMU.

Altri comandi utili forniti da MokoMakefile sono:

- $ make flash-qemu-official
- $ make flash-qemu-local

Tali comandi si occupano di effettuare il flash delle immagini (ultima versione disponibile su internet o locale) sull'emulatore.

- $ make run-qemu
- $ make run-qemu-snapshot

Tali comandi mandano in esecuzione l'emulatore

3.9 Problemi riscontrati e noti

Durante la creazione dell'immagine si sono riscontrati diversi problemi prevalentemente dovuti a:

- aggiornamenti quotidiani dei repository

- aggiornamenti periodici delle guide e delle procedure
- problemi di crc e di download dei file
- architettura a 64bit non completamente supportata

Per quanto riguarda i primi due punti, circa ogni giorno vengono aggiornati i sorgenti nei repository remoti e MokoMakefile tenterà di compilarne sempre l'ultima versione per ciascuno. Può capitare che alcuni di essi non funzionino correttamente o contengano errori. In questi casi E' necessario attendere e aspettare che venga rilevato e corretto il problema.

Vengono di seguito riportati alcuni trucchi e consigli, derivati dall'esperienza e dalla lettura del manuale e del MokoMakefile stesso.

3.9.1 Ricompilazione specifica di un pacchetto

Può capitare che un pacchetto non venga correttamente compilato per errori nel crc oppure per interruzioni causate dall'utente o altri motivi. In questi casi è meglio cancellare i sorgenti e ricompilare da zero il software in questione.
Tramite MokoMakefile è possibile fare ciò tramite i seguenti comandi:

```
$ make clean-package-<PACKAGE>
$ make build-package-<PACKAGE>
```

Un metodo analogo è quello di utilizzare bitbake con i seguenti comandi:

```
$ rm sources/<PACKAGE>*
$ cd build
$ . ../setup-env
$ bitbake -crebuild <package>
```

In questo modo è anche possibile compilare un software bypassando MokoMakefile andando ad agire manualmente. L'equivalente di:

```
$ make openmoko-devel-image
```

si può ottenere come segue:

```
$ cd build
$ . ../setup-env
$ bitbake openmoko-devel-image
```

3.9.2 Segnalazione problemi

Nel caso i problemi non vengano risolti con i metodi sopra citati si consigla di segnalarlo collegandosi sul server `irc://irc.freenode.net` canale `#openmoko`.

Capitolo 4

Toolchain

E' possibile utilizzare una toolchain per sviluppare software da installare direttamente sull'openmoko senza dover necessariamente ricompilare tutta l'immagine.

4.1 Download ed Installazione

La toolchain è fornita in 3 diversi modi:

- binari per x86
- binari per x86_64
- pacchetto bitbake

Per scaricare i binari si può procedere come segue:

```
$ wget http://downloads.openmoko.org/toolchains/openmoko-x86_64-arm-linux-gnueabi-too
oppure
$ wget http://downloads.openmoko.org/toolchains/openmoko-i686-arm-linux-gnueabi-toolc
```

Una volta scaricata l'immagine la si può estrarre e spostare in /usr/local con i comandi:

```
$ tar xfvj openmoko*tar.bz2
$ sudo mv usr/local/openmoko /usr/local
```

A questo punto il sistema è pronto per compilare software e creare pacchetti ipk.

4.1.1 compilazione toolchain

E' possibile compilare la toolchain a partire dai sorgenti tramite bitbake o anche tramite MokoMakefile stesso:

```
$ make openmoko-toolchain
oppure
$ cd build/
$ . ../setup-env
$ bitbake meta-toolchain-openmoko
```

Questo creerà un file compresso della toolchain nella directory $OMDIR/build/tmp/deploy/sdk appositamente compilato per il sistema corrente.

4.1.2 pacchettizzare la toolchain

In sistemi Debian-based è possibile creare un file .deb della toolchain. Ciò agevolerebbe l'upgrade e la disinstallazione di essa. Per farlo sono necessari i seguenti comandi:

```
$ bunzip2 openmoko-x86_64-arm-linux-gnueabi-toolchain.tar.bz2
$ gzip openmoko-x86_64-arm-linux-gnueabi-toolchain.tar
$ fakeroot alien -d openmoko-x86_64-arm-linux-gnueabi-toolchain.tar.gz
```

4.2 Compilazione di un Software

Si consideri un generico software distribuito in sorgenti che necessita la classica procedura di compilazione:

```
$ ./configure
$ make
# make install
```

Per poter cross-compilare il software al fine di renderlo eseguibile sul neo1973 (arm) è necessario modificare il Makefile fornito con i sorgenti per fargli utilizzare il compilatore e le librerie della toolchain.

Per fare ciò viene fornito un software, **om-conf** che analizza il configure, lo esegue e lo modifica opportunamente.

Fatto ciò si procede con l'esecuzione del make per la compilazione. Per l'installazione invece è necessario utilizzare un software fornito dalla toolchain, **om-make-ipkg**, che si occuperà di trasformare il programma appena compilato in un pacchetto ipk da poter installare direttamente sul telefono stesso.

Vediamo ora di eseguire la procedura su un software di prova fornito insieme alla toolchain. I sorgenti sono disponibili nella directory:

```
/usr/local/openmoko/source/openmoko-sample2/
```

La procedura per ottenere un ipk è la seguente:

```
$ . /usr/local/openmoko/arm/setup-env
$ cp -a /usr/local/openmoko/source/openmoko-sample2/ ~
$ cd ~/openmoko-sample2
$ om-conf . <PARAMETRI_DI_CONFIGURE>
$ make
$ om-make-ipkg
```

Fatto questo avremo il nostro file openmoko-sample2_0.1_armv4t.ipk nella working directory. E' possibile inserire nel pacchetto informazioni riguardanti l'autore, la versione e una descrizione. Per fare ciò è necessario utilizzare un ipkg control file.

4.3 Caricamento e Installazione di un pacchetto

Per caricare il pacchetto appena realizzato sul telefono, è necessario utilizzare il cavo usb. Tramite il modulo del kernel usbnet è possibile par passare il protocollo ip sopra a un link usb. Una volta caricato il modulo e collegato il telefono, sul computer locale sarà disponibile l'interfaccia di rete usb0.

Di default il telefono assume l'indirizzo 192.168.0.202 e si aspetta un default gateway e dns sull'interfaccia usb0 all'indirizzo 192.168.0.200.

E' presente un server ssh già configurato per poter caricare file. L'utente è root e non richiede alcuna password. Ecco i comandi per caricare e installare il paccketto:

```
$ ifconfig usb0 192.168.0.200
$ scp openmoko-sample2_0.1_armv4t.ipk root@192.168.0.202:/home/root/
$ ssh root@192.168.0.202 opkg install /home/root/openmoko-sample2_0.1_armv4t.ipk
```

Bibliografia