OpenEmbedded

OpenEmbedded è un sistema di build che permette di crosscompilare distribuzioni per macchine con risorse limitate, specialmente se di architetture diverse dall'x86.

Usi comuni

L'applicazione tipica di OpenEmbedded è la generazione di sistemi Linux per dispositivi embedded in grado di sfruttare la presenza di un sistema completo; tipicamente palmari, cellulari o dispositivi di rete.

Rispetto ad una distribuzione tradizionale pone l'enfasi sull'adattamento dei programmi a dispositivi di capacità ridotte. Contrariamente alla maggior parte dei sistemi di build per sistemi embedded però non crea immagini composte da un kernel e pochi programmi compilati staticamente, ma una vera e propria distribuzione completa e dotata anche di un sistema di gestione dei pacchetti.

Componenti

Metadati

I metadati di openembedded sono distribuiti tramite un repository git e sono composti da file di testo che descrivono la configurazione ed il processo di crosscompilazione.

```
classes  contrib      files        packages  removal.txt
conf     COPYING.MIT  MAINTAINERS  README    site
```

Macchine

Con *machine* si intende un dispositivo supportato da OpenEmbedded; per ogni dispositivo esiste un file in `conf/machine` che definisce le variabili che descrivono l'architettura e le caratteristiche del dispositivo.

Esempi di macchine supportate:

beagleboard un computer su singola scheda basato su OMAP 3530, <http://beagleboard.org>_

i586-generic un generico PC con processore pentium o successivi, utile per mini-PC

nslu2be e nslu2le NSLU: un NAS della Linksys facilmente riflashabile

omap3-pandora una console di gioco per homebrew

om-gta02 Neo FreeRunner: il cellulare libero distribuito con il sistema OpenMoko

x86 generico PC con architettura x86

Tipicamente, per supportare un nuovo modello di hardware si copia la macchina piu' simile, e la si adatta alle proprie esigenze.

Distribuzioni

Una distribuzione definisce le versioni dei programmi che verranno installati e la configurazione del sistema, tramite la definizione di opportune variabili in un file in `conf/distro`.

Esistono alcune distribuzioni sviluppate indipendentemente, come OpenMoko o Poky, che però mantengono anche un branch semiindipendente e personalizzato dei metadati. La distribuzione standard è invece Ångström, disponibile in una versione stabile ed una di sviluppo, ciascuna con un suo branch dei metadati sul repository standard.

Ricette

L'altra componente principale dei metadati sono le ricette, contenute nella directory `packages`, che contengono le istruzioni per la crosscompilazione di specifici programmi e la creazione di pacchetti ed immagini.

Pacchetti

La maggior parte delle ricette corrisponde ad un programma e permette di crosscompilarne i relativi pacchetti.

Ad esempio la ricetta dell'editor *nano* inizia con le informazioni che andranno nel pacchetto generato, comprese le dipendenze:

```
DESCRIPTION = "GNU nano (Nano's ANOther editor, or \
Not ANOther editor) is an enhanced clone of the \
Pico text editor."
HOMEPAGE = "http://www.nano-editor.org/"
LICENSE = "GPLv2"
SECTION = "console/utils"
DEPENDS = "ncurses"
```

prosegue con l'indirizzo da cui prendere i sorgenti ed eventuali patch:

```
SRC_URI = "http://www.nano-editor.org/dist/v2.0/nano-${PV}.tar.gz \
           file://glib.m4"
```

quindi dice che il pacchetto utilizza autotools per la compilazione, specificando delle opzioni aggiuntive per il comando `./configure`:

```
inherit autotools

# only 16K more to get everything but the kitchen sink
EXTRA_OECONF = "--enable-all"
```

infine definisce delle azioni aggiuntive da effettuare nel corso della crosscompilazione:

```
do_configure_prepend () {
        install -m 0644 ${WORKDIR}/glib.m4 m4/
}
```

Task

I task sono delle ricette che creano metapacchetti, in modo da semplificare l'installazione di numerosi pacchetti correlati tramite il sistema di gestione delle dipendenze. Sono contenuti in `packages/tasks`

Alcuni task particolarmente utili sono:

task-native-sdk per aggiungere l'SDK nativa sul dispositivo

task-proper-tools per inserire le versioni standard dei programmi che generalmente sui sistemi di dimensioni limitate vengono sostituiti da busybox; ad esempio vim, sed, grep, eccetera

Immagini

Le ricette per le immagini, contenute in `packages/images` descrivono come creare un'immagine con un sistema completo che possa essere caricata ed usata sul dispositivo.

Configurazione locale

Infine, è utile citare il file di configurazione locale, al di fuori del tree dei metadati di openembedded, con il quale si definiscono le directory da usare, la macchina e la distribuzione ed eventuali opzioni di build.

Un esempio viene distribuito in `conf/local.conf.sample`, come descritto nella documentazione sull'installazione.

Bitbake

Bitbake è il programma, scritto in python, che gestisce scaricamento, compliazione e pacchettizzazione dei programmi, un po' come emerge in gentoo.

Il suo uso comune è semplicemente:

```
bitbake <ricetta>
```

dove <ricetta> è il nome di una ricetta di un programma, un task o un'immagine; ad esempio:

```
bitbake nano
bitbake task-proper-tools
bitbake x11-image
```

A quel punto, se tutto funziona correttamente, bitbake provvederà autonomamente a svolgere tutti i passaggi necessari per generare quanto richiesto.

Task

La creazione dei pacchetti è suddivisa in passaggi autonomi, detti task; l'output di openembedded mostra il task in corso ed in caso di interruzione bitbake riparte sempre sfruttando i task già completati.

Alcuni task importanti sono i seguenti.

fetch: scarica i sorgenti del pacchetto.

unpack: estrae i sorgenti dagli archivi.

patch: applica eventuali patch ai sorgenti.

configure: configura i sorgenti; se esiste uno script ./configure viene lanciato in questo task.

compile: compila il pacchetto.

populate_staging: installa eventuali librerie compilate per il target nell'area di "staging" sull'host, perche' possano essere usate durante la compilazione di altri pacchetti.

install: installa i programmi in una directory dalla quale verranno creati i pacchetti finali.

package_write: crea il pacchetto del programma.

Un ulteriore task, clean, permette di cancellare tutto quello che è stato fatto per una ricetta, in modo da ricominciare da capo; può ovviamente solo essere lanciato singolarmente con:

```
bitbake -c clean <ricetta>
```

Output

Anziché creare un'immagine statica del filesystem da caricare sui dispositivi, come è consueto per i sistemi di crosscompilazione, OpenEmbedded usa un'approcio ibrido e crea dei pacchetti binari come quelli delle comuni distribuzioni, che vengono poi installati in un'immagine *flashabile*.

Immagini

Le immagini generate da bitbake si trovano nella directory `$TMP/deploy/<librerie c>/images/<architettura>/` e possono essere degli archivi tar eventualmente compressi contenenti i file che compongono il sistema oppure delle vere e proprie immagini di filesystem di dimensioni predefinite per la macchina, che possono essere copiate sulla memoria del dispositivo target; in quest'ultimo caso la scelta è tra jffs2, cramfs squashfs ed ext(2|3)

Pacchetti

I pacchetti generati da bitbake per i vari programmi sono nel formato ipk/opk, ispirati ai .deb, ma ridotti per adattarsi alle esigenze dei sistemi embedded; vengono gestiti dal programma opkg ed è possibile creare dei veri e propri repository online, usabili esattamente come un repository tradizionale.

Si trovano nella directory `$TMP/deploy/<librerie c>/ipk/<architettura>/`.

La distinzione tra ipk e opk è storica: opk ed opkg sono la nuova versione, dove avviene attualmente lo sviluppo di ipk ed ipkg rispettivamente.

Link utili

OpenEmbedded Wiki Il sito principale di OpenEmbedded

Getting Started Istruzioni su come installare OpenEmbedded sul proprio sistema host

repository Il repository dove sono conservati i metadati

Distribuzione Ångström La distribuzione standard di OpenEmbedded

Questo documento

Authors: Elena "of Valhalla" Grandi <valhalla@lifolab.org>

Version: 2009-01-19