KAEILOS E OPENEMBEDDED LINUX

Cavallini, Marco, KOAN s.a.s., Bergamo - Italia, m.cavallini@koansoftware.com

Abstract – KaeilOS e OpenEmbedded per generare sistemi linux embedded

KaeilOS è una distribuzione linux embedded Open Source creata dalla società KOAN per fornire un sistema che permettesse di creare una distribuzione linux embedded il più comodamente possibile. Per ottenere ciò il sistema KaeilOS è stato inizialmente basato su un tool per la generazione automatica di target Linux denominato PTXdist. Recentemente KaeilOS è confluito nel progetto OpenEmbedded (OE).

Sebbene esistano progetti o soluzioni che forniscono pacchetti già opportunamente cross-compilati semplificando la vita dello sviluppatore, le soluzioni automatizzate come KaeilOS e OpenEmbedded hanno scelto fin dall'inizio la strada di costruire tutto da zero "from scratch" per permettere all'utente di avere il completo controllo dell'intero ciclo di sviluppo del sistema finale.

L'approccio dei sistemi che forniscono un tale livello di automazione viene talvolta ritenuto troppo pesante e complesso, tuttavia contrariamente ad altri sistemi ed in particolare al classico linux fai da te "DIY" risulta un importante strumento per aumentare la produttività ed il time-to-market per i nuovi utenti e tutte le persone/clienti che si approcciano per la prima volta a Linux embedded, collocando KaeilOS, ed i progetti su cui esso si basa, alla pari di blasonati e costosi strumenti di sviluppo proprietari.

Verranno presentati brevemente alcuni tools alternativi a OpenEmbedded ma ritenuti meno interessanti come base per KaeilOS, quali PTXdist, Poky e Scratchbox.

OpenEmbedded (OE) è un ambiente di sviluppo che permette di cross-compilare sul proprio PC applicazioni per sistemi linux embedded. E' utilizzato per la generazione di distribuzioni linux come Ångström, OpenZaurus, OpenMoko e recentemente per KaeilOS. ***OpenEmbedded*** *è un repository di metadati che vengono elaborati dal parser/compilatore di metadati* ***bitbake*** *al fine di generare un'immagine di linux da zero.*

Le differenze e peculiarità di KaeilOS rispetto alla sua distribuzione di riferimento Ångström sono la particolare attenzione alle problematiche legate all'ambito industriale e quindi l'affidabilità ed efficienza e il supporto a lungo termine, tutte caratteristiche non pienamente disponibili in Ångström e OpenEmbedded.

E' possibile usare KaeilOS per compilare il software che si intende installare sul proprio dispositivo embedded.

OpenEmbedded si basa principalmente sui seguenti componenti:

- ***bitbake****, un tool scritto in python che assomiglia molto ad emerge di gentoo, è in pratica il "build manager" di openembedded*
- ***git****, un sistema software per il controllo di versione distribuito, originariamente creato da Linus Torvalds per gestire le modifiche al kernel di linux.*
- *i files* ***.bb****, chiamati 'ricette', ovvero dei file di testo che contengono le istruzioni per bitbake, uno per ogni pacchetto, o per ogni compito supplementare da eseguire*
- *il proprio* ***local.conf****, un file di configurazione che dice a bitbake per quale dispositivo o distribuzione vogliamo compilare il software*

Verranno dunque trattate le problematiche legate alla gestione, installazione e manutenzione del progetto KaeilOS/OE e dei componenti da esso utilizzati: bitbake e git.

Parole Chiave*: KOAN, KaeilOS, Klinux, Linux, Embedded, OpenEmbedded, bitbake, git, PTXdist, Scratchbox, ARM, x86*

1 INTRODUZIONE

1.1 Approccio a Linux embedded

Con 'Embedded Linux' ci si riferisce a quell'insieme di distribuzioni concepite per essere utilizzate su sistemi embedded. Le caratteristiche principali di tali sistemi impongono dei vincoli molto severi al sistema operativo in termini di memoria flash occupata, memoria centrale necessaria, tempi di avvio brevi.

Sebbene la maggioranza degli sviluppatori sia in grado di utilizzare le normali distribuzioni Linux come SuSE, RedHat, Mandrake o Debian come base per le loro applicazioni, per i sistemi embedded le cose sono diverse. A causa delle risorse limitate di cui dispongono in genere questi sistemi, le distribuzioni devono essere di piccole dimensioni e dovrebbero contenere solo le cose che sono necessarie per l'applicazione. Inoltre, il sistema deve essere verificabile e riproducibile in quanto di solito gli sviluppatori embedded desiderano sapere che cosa c'è dentro i propri sistemi, in particolare se essi devono mantenere il loro software per un lungo periodo come ad esempio i 10-15 anni di ciclo di vita dei prodotti per applicazioni di automazione. L'approccio più comunemente adottato per generare un sistema linux embedded può essere uno dei seguenti:

- il classico Do It Yourself (DIY) Linux From Scratch, procedura complessa e per utenti esperti.
- la riduzione (downscaling) di una delle distribuzioni disponibili al momento (Debian, Suse, RedHat), laboriosa e poco efficiente.
- l'utilizzo di una distribuzione embedded già pronta (etlinux, Midori, PeeWeeLinux, Familiar), tutte piuttosto vecchie o con limitazioni in termini di architetture supportate e pacchetti disponibili e quindi poco adatte a sistemi embedded.
- l'adozione di un tool per la generazione automatica del sistema target tra quelli elencati di seguito.

Linux From Scratch consiste in un modo per installare un sistema Linux funzionante attraverso la costruzione manuale e configurazione di tutti i suoi componenti. Tale processo è molto più laborioso rispetto all'installazione di una distribuzione pre-costruita. L'idea di base è che installare i singoli pacchetti uno per uno porterà ad una comprensione dei meccanismi interni di un sistema Linux funzionante. Inoltre, ovviamente, compilare tutto il software specificamente per la piattaforma ed architettura su cui verrà eseguito tende a far risultare i programmi più leggeri e veloci. Infine, è più facile personalizzare i pacchetti installati quando ognuno di essi è stato installato manualmente.

Il concetto sviluppato dai tool per la generazione automatica del sistema target è di prendere del software e creare qualcosa che si può eseguire su un altro dispositivo. Ciò comporta il download del codice sorgente, la compilazione, la creazione di pacchetti (come .deb .rpm .ipk) e la creazione dell'immagine di boot che può scritta sul dispositivo di storage del sistema target.

Le difficoltà di cross-compilazione e la varietà di dispositivi che possono essere supportati, porta un a notevole complessità in più rispetto a quella che si può trovare in una tipica distribuzione desktop (dove la cross-compilazione non è necessaria), da ciò è nata l'esigenza di poter disporre di sistemi che raggruppassero tutto ciò che serve in un'unica soluzione. I più noti tool per la generazione automatica di sistemi Linux embedded sono i seguenti:

- **buildroot**, interessante ma poco attivo e con frontend di configurazione complesso
- **PTXdist**, derivato da buildroot e con un'ampio supporto di pacchetti e architetture
- **OpenEmbedded**, molto attivo, basato su bitbake e git
- **Poky** (basato su OpenEmbedded)
- **Scratchbox**, un toolkit per la cross compilazione

- **uClinux**, enorme e complesso da mantenere o nell'aggiungere pacchetti
- **emDebian**, versione embedded di Debian, non supportato e basato su CML2 abbandonato

1.2 La visione di KOAN

Tutti gli sforzi di KOAN nel mantenere la propria distribuzione KaeilOS embedded sono mirati al raggiungimento di un obiettivo ambizioso: fornire ai nuovi utenti/clienti che si approcciano per la prima volta a Linux embedded un sistema guidato per la generazione del sistema target, permettendo all'utente di potersi concentrare solo sull'applicativo o le attività strettamente legate al proprio core-business. Questo genere di soluzioni risulta quindi essere un importante strumento per aumentare la produttività ed il time-to-market, permettendo di collocare **KaeilOS**, ed i progetti su cui esso si basa, alla pari di blasonati e costosi strumenti di sviluppo proprietari per il mercato software embedded (Montavista, VxWorks, QNX, Windows CE Platform Builder, ecc...).
Già dalla fine del 1999 la società KOAN era in grado di offrire questo tipo di prodotto (denominato **Klinux**) basato su sistemi e pacchetti completamente Open Source ed è stata una delle prime aziende in Italia ad offrire consulenza per sistemi Linux embedded.

2 TOOL PER LA GENERAZIONE DI LINUX EMBEDDED

Una breve descrizione dei tools presi in esame da KOAN prima di giungere alla scelta di OpenEmbedded come sistema su cui basare il proprio lavoro e il supporto tecnico commerciale.

2.1 PTXdist

PTXdist è un sistema per la generazione di sistemi Linux embedded rilasciato in licenza GNU/GPL e sviluppato dall'azienda tedesca Pengutronix. PTXdist è basato sul sistema di configurazione Kconfig del kernel di linux ed è stato il sistema su cui KOAN ha basato **Klinux** alla fine del 1999 (rinominandolo **KaeilOS** nel 2006 per ragioni di Trademark).

PTXdist definisce tramite degli switch di configurazione che devono essere selezionati con un'interfaccia semigrafica a caratteri basata su ncurses (come menuconfig del kernel). Questi switch definiscono quali opzioni devono essere considerate, quali modifiche devono essere applicate per configurare e successivamente compilare i pacchetti che andranno a comporre il sistema finale.

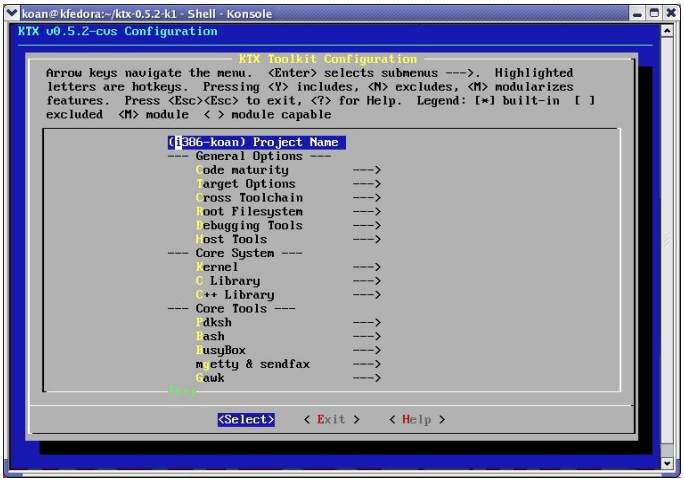

Figura 1. Una videata del configuratore di PTXdist

2.2 Poky linux

Poky è un progetto derivato da OpenEmbedded che però ha delle particolarità ben precise pur rimanendone molto simile nello sviluppo, infatti, conserva la scelta di bitbake come sistema per la gestione di dipendenze e compilazione di pacchetti, ma ha un albero di directorty completamente diverso. Poky era supportato dall'azienda inglese Opened-Hand, ora acquisita da Intel. Il progetto Poky ha risentito gravemente del cambio di proprietà dell'azienda mantainer e si trova ora in una situazione di stallo. Ciò ha provocato la migrazione a OpenEmbedded di molti utenti di Poky, così come diversi pacchetti e ricette disponibili solo in Poky sono stati ora importati in OE.

Le principali caratteristiche di Poky sono:

- Un sistema completo di Linux kernel, X11, Matchbox, GTK+, Pimlico, Clutter, e altri applicativi GNOME, oltre ad una completa piattaforma di sviluppo.
- Un sottoinsieme di OpenEmbedded focalizzato e stabile che può essere usato facilmente.
- Pieno supporto ad un'ampia gamma di hardware x86 e ARM.

Figura 2. Una videata di Sato con Poky linux

Poky è principalmente una piattaforma per sviluppare un'immagine di filesystem basata su software opensource come il server X Kdrive, il window manager Matchbox, il toolkit GTK+. L'immagine del file system può essere generata per diversi tipi di device, ma è possibile anche usare come target una macchina virtuale QEMU che emula hardware x86 e ARM.

Un importante componente integrato con Poky è Sato, un'interfaccia grafica GNOME ottimizzata per dispositivi mobile. Questa interfaccia è stata sviluppata per lavorare bene con schermi ad alta risoluzione e ristretta dimensione, come quelli che spesso si trovano nei PDA. Un'altra caratteristica di Poky è quella di avere un eccellente supporto per un numero limitato di architetture hardware.

2.3 Scratchbox

Scratchbox è un toolkit di cross-compilazione ideato per rendere più semplice lo sviluppo di applicazioni Linux embedded. Mette a disposizione un set completo per integrare e cross-compilare una intera distribuzione Linux. Scratchbox ha un'area (test environment) dove viene verificato il corretto funzionamento dei suoi componenti in modo da assicurare l'affidabililtà d'uso di determinate versioni di librerie, header e applicativi.

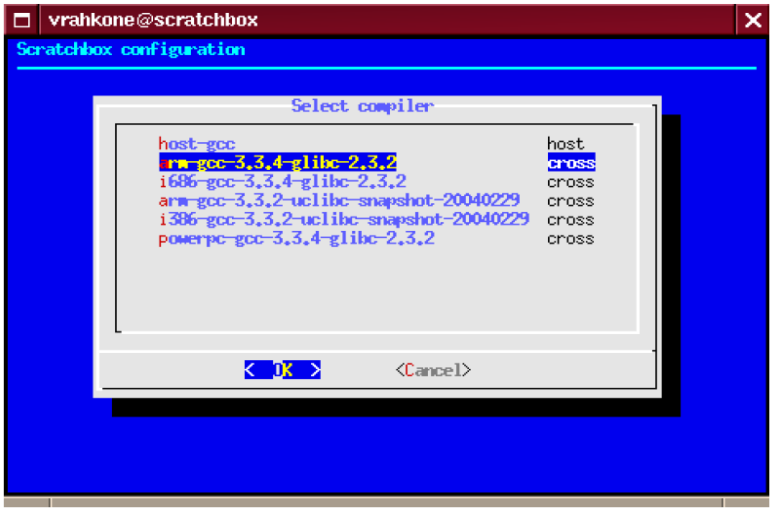

Figura 3. Una videata del configuratore di Scratchbox

Scratchbox permette di cross-compilare pacchetti software per x86 e ARM. L'ambiente è impostato come una shell dalla quale è possibile già dall'installazione eseguire i binari compilati. Questo è possibile grazie al supporto dell'emulatore Qemu che implementa (oltre che una macchina hardware) lo strato di compatibilità fra architetture diverse. E' possibile, quindi, tramite la shell di scratchbox compilare ed eseguire ad esempio binari ARM su architettura x86. E' possibile installare pacchetto per pacchetto e creare un filesystem personalizzato. Scratchbox permette, inoltre, di agganciarsi ai repository ufficiali Debian (anche Slackware) e installare un sistema da zero. E' possibile utilizzare il comando apt-get e iniziare a costruire il proprio filesystem con software compilato da debian per le architetture x86 e arm.

Scratchbox è utilizzato da Nokia che ha sviluppato il progetto Maemo. Maemo ha un Xserver e un ambiente grafico basato su gtk interamente emulabile grazie alla shell scratchbox (tramite Qemu). Scratchbox inoltre mette a disposizione diverse toolchain che comprendono gcc ottimizzato per diverse CPU e uClibc

3 OPENEMBEDDED

Poiché KaeilOS si basa su OpenEmbedded (OE), tutto il funzionamento ed i files di configurazione fondamentali per il sistema è essenzialmente quello di OE, solo alcuni aspetti e configurazioni sono specifiche di KaeilOS. Per questo motivo verrà dapprima presa in esame l'architettura di OE in modo da avere un'introduzione ai suoi concetti essenziali.

3.1 Nascita di OpenEmbedded

Il progetto OpenEmbedded è stato creato originariamente nel 2003 da un gruppo di sviluppatori del progetto OpenZaurus ed in particolare da Chris Larson (overall architecture), Holger Schurig (first implementation), e Michael Lauer (first loads of packages and classes).

Altre distribuzioni hanno iniziato ad adottare OE: Unslug, OpenSimpad, GPE Phone Edition, Ångström, OpenMoko e recentemente anche **KaeilOS**. Ognuna di queste distribuzioni apporta il proprio bagaglio di esperienze e di specifiche esigenze al progetto OE, rendendolo una vera e propria fucina di pacchetti e architetture supportate.

3.2 Introduzione

Il concetto sviluppato da OpenEmbedded, come per altri sistemi analoghi, è di prendere del software e creare qualcosa che si può eseguire su un altro dispositivo. Ciò comporta il download del codice sorgente, la compilazione, la creazione di pacchetti (come .deb .rpm .ipk) e la creazione dell'immagine di boot che può scritta sul dispositivo di storage del sistema target.

Le difficoltà di cross-compilazione e la varietà di dispositivi che possono essere supportate, porta un po' complessità in più in una distribuzione basata su OpenEmbedded che quella che si può trovare in una tipica distribuzione desktop (dove la cross-compilazione non è necessaria).

Una parte importante di OpenEmbedded riguarda la compilazione di codice sorgente per vari progetti. Per ogni progetto viene generalmente richiesta la stessa sequenza dei seguenti compiti:

- Scaricare il codice sorgente, e relativi file di supporto (come initscripts);
- Estrarre il codice sorgente e applicare tutte le patch che possono essere necessarie;
- Configurare il software, se necessario (come si fa eseguendo lo script 'configure');
- Compilare il tutto;
- Pacchettizzare tutti i file in uno dei formati disponibili, come .deb o .rpm o .ipk, pronti per l'installazione.

Non c'è niente di particolarmente, insolito in questo processo di costruzione del sistema, quando i pacchetti devono essere installati sulla stessa macchina dove sono compilati. Nel caso di sistemi embedded, ciò che rende difficile queste procedure sono i seguenti aspetti:

- Cross-compilazione: cross-compilare è difficile, e un molto software non ha alcun supporto per la cross-compilazione - tutti i pacchetti inclusi nel OE sono cross-compilati;
- Target e l'Host sono diversi: questo significa che non è possibile compilare un programma e poi eseguirlo – esso è compilato per funzionare con il sistema target, non sul sistema di Host di compilazione. Molti software cercano di costruire e gestire piccole applicazioni di supporto alla compilazione stessa e questo non funziona quando si cross-compila.

- Toolchains (compilatore, linker, ecc...) sono spesso difficili da compilare. Le cross toolchains sono ancora più difficili. In genere si tende a scaricare una toolchain fatta da qualcun altro - ma non quando si utilizza OE. In OE tutta la toolchain è creata come parte del processo di generazione del sistema finale. Ciò può rendere le cose inizialmente più lunghe e può rendere più difficile iniziare, ma rende più facile applicare le patch e testare le modifiche alla toolchain.

Naturalmente in OE c'è molto di più che la semplice compilazione dei pacchetti, alcune delle caratteristiche che supporta comprendono:

- Supporto per entrambe glibc e uclibc;
- Supporto per la generazione per diversi dispositivi target da un'unica base di codice;
- Costruire automaticamente tutto ciò che è necessario per compilare e/o eseguire il pacchetto (compilare ed eseguire le sue dipendenze);
- Creazione di immagini disco flash adottando uno qualsiasi dei tipi supportati (jffs2, ext2, gz, squashfs, ecc...) per fare il boot direttamente sul dispositivo target;
- Supporto per vari formati di pacchettizzazione;
- Costruzione automatica di tutti gli strumenti di cross-compilazione necessari;
- Supporto per pacchetti "nativi" che sono costruiti per il computer host e non per il target e utilizzati durante il processo di compilazione;

3.3 Configurazione

Configurazione di base comprende elementi come ad esempio l'indicazione su dove possono essere situati vari files e dove dovranno essere messi i files generati, specificando ad esempio qual'è l'hardware preso in esame come target e quali caratteristiche si desidera avere incluse nell'immagine finale. La configurazione di OE è distribuita in un albero di directory organizzato nel modo seguente:

```
openembedded/conf
|-- machine
|-- distro
|-- bitbake.conf
`-- local.conf      *KaeilOS uses a different location
```

Analizziamo in dettaglio il contenuto di ognuna di questa directory.

```
conf/machine
```

Questa directory contiene le informazioni di configurazione della macchina. E' necessario specificare in questa directory un file di configurazione per ogni dispositivo 'target' fisico. Il file deve descrivere i vari aspetti del dispositivo, come ad esempio l'architettura, le caratteristiche hardware (dispone di USB? di una tastiera? ecc...), il tipo di immagine flash o del disco, le impostazioni della console seriale, ecc... Se si vuole aggiungere il supporto per un nuovo dispositivo si deve creare un nuovo file di configurazione per il dispositivo in questa directory.

```
conf/distro
```

Questa directory contiene i file di distribuzione. Una distribuzione decide come le varie attività sono gestite nell'immagine finale, ad esempio come dovrà essere configurata la rete, se i dispositivi USB saranno supportati, quale sistema di pacchettizzazione deve essere usato, quale tipo di libc utilizzare, ecc...

```
conf/bitbake.conf
```

Questo è il file di configurazione principale per **bitbake**. Questo file non deve essere modificato, ma è utile prenderlo in considerazione dal momento che dichiara il maggior numero delle variabili predefinite utilizzate da OE e controlla molte delle funzionalità di base fornite da OE.

```
conf/local.conf
```

Questo il file di configurazione specifico dell'utente finale. Questo file deve essere copiato e modificato, ed è usato per indicare le varie directory di lavoro, la macchina per la compilazione e la distribuzione da utilizzare, nel nostro caso sarà indicata dalla stringa **"kaeilos"**.

3.4 Il branch 'stable'

A seguito delle richieste da parte di molte aziende che hanno fatto di OpenEmbedded il sistema di sviluppo per le proprie architetture, da Marzo 2009 è stata introdotta un'area di testing e validazione (stable/2009) dove viene verificato il corretto funzionamento dei suoi componenti in modo da assicurare l'affidabililtà d'uso di determinate versioni di librerie, header e applicativi. Il branch stable è utilizzato da KeilOS come repository di default.

4 IL PROGETTO KAEILOS

Poiché KaeilOS è basato su OpenEmbedded (OE), i riferimenti a KaeilOS/OE che seguiranno possono essere applicati anche ad OE.

4.1 Caratteristiche di KaeilOS

KaeilOS/OE è un progetto giovane, e necessita ancora di un po' di tempo per poter raggiungere i principali obiettivi che si è prefissato, pertanto al momento KaeilOS utilizza Ångström come distribuzione di riferimento tra quelle presenti in OE.

L'intenzione di KOAN è di portare le stesse peculiarità di KaeilOS 3.0 basato su PTXdist anche nella nuova versione 4.0 basata su OE.

Queste caratteristiche mirano principalmente ad ottenere le seguenti caratteristiche:

- boot più rapido
- supporto principale del filesystem in ramdisk
- eliminazione della gestione dinamica dei dispositivi con udev, a favore di mdev
- eliminazione del supporto DHCP
- eliminazione dei servizi (demoni) non espressamente richiesti in fase di configurazione
- possibilità di estendere il sistema con patch Real-Time (Xenomai o RTAI)

Sebbene le caratteristiche che KaeilOS vuole introdurre possano sembrare limitative, è tuttavia necessario considerare che in un sistema embedded normalmente le caratteristiche hardware sono praticamente statiche a differenza di quanto può accadere in un sistema desktop e quindi non esiste la necessità di adattarsi ai dispositivi esterni che possono eventualmente venire collegati.

4.2 Componenti del sistema KaeilOS

Il progetto KaeilOS offre componenti, tools e servizi virtualmente comuni a tutti i progetti Linux embedded, racchiudendoli sotto un unico prodotto.

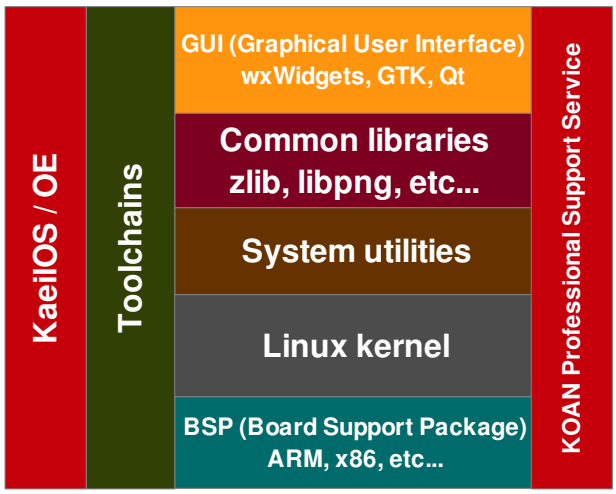

Figura 4. Schema a blocchi dei componenti e dei servizi offerti da KaeilOS

5 INSTALLAZIONE DI KAEILOS

Per installare KaeilOS è necessario seguire alcuni passi fondamentali costantemente aggiornati al seguente indirizzo web: http://www.kaeilos.com/?q=download

Le indicazioni sono semplificate al massimo e seguite pedissequemente permettono all'utente non pratico di linux di ottenere facilmente un sistema di sviluppo pronto per l'uso e uniformato secondo lo standard previsto da KOAN per un rapido supporto tecnico. Ovviamente gli utenti più esperti potranno personalizzare il sistema secondo le proprie specifiche necessità o seguire le indicazioni originali di OE: http://wiki.openembedded.net/index.php/Getting_Started

5.1 Sistema hardware per lo sviluppo

Per la compilazione di KaeilOS/OE è consigliato un sistema il più potente possibile come ad esempio quello suggerito:

- CPU Quad core
- Motherboard con chipset Intel (o con elevate prestazioni di I/O sul bus)
- 4GB RAM
- 320GB HD (elevata cache e RPM daranno risultati migliori)
- Debian Lenny 5.0.1 (amd64)

5.2 Pacchetti richiesti

Per compilare un sistema basato su KaeilOS/OE occorre installare il seguente software:

```
sudo apt-get install ccache sed wget cvs subversion
sudo apt-get install coreutils unzip texinfo libsdl1.2-dev docbook-utils
sudo apt-get install gawk python-pysqlite2
sudo apt-get install libxml2-utils xmlto
sudo apt-get install git-core
sudo apt-get install help2man diffstat texi2html
sudo apt-get install qemu          (if you want to emulate a target system)
sudo apt-get install python-psyco  (only for i386, not for amd64)
sudo apt-get install texinfo       (required by md5)
sudo apt-get install docbook
```

5.3 Preparazione della macchina di sviluppo (Host)

Sulla macchina Host di sviluppo, suggeriamo di installare Debian Lenny, ma qualunque distribuzione è adatta allo scopo. Per particolari dettagli specifici si rimanda alle indicazioni riportate sul sito di OE: http://wiki.openembedded.net/index.php/OEandYourDistro

5.4 Creazione dell'albero di sviluppo

Preparare una directory nella home dove ospitare l'albero di sviluppo

```
cd /home
sudo mkdir koan
sudo chown `whoami`.`whoami` /home/koan
cd /home/koan
```

5.5 Scaricare bitbake

quindi scaricare l'ultima versione di **bitbake**

```
mkdir /home/koan/devel
cd /home/koan/devel
svn co svn://svn.berlios.de/bitbake/branches/bitbake-1.8/ bitbake
```

Si raccomanda di usare bitbake senza installarlo nel sistema globale ma esclusivamente nelle directory dell'ambiente di sviluppo bitbake. Si raccomanda inoltre di non utilizzare la versione di bitbake fornita con la propria distribuzione.

Bitbake è costantemente aggiornato, pertanto si consiglia di aggiornarlo di tanto in tanto con i seguenti comandi

```
cd /home/koan/devel/bitbake
svn update
```

5.6 Scaricare OpenEmbedded

Scaricare Openembedded dal repository ufficiale

```
cd /home/koan/devel
git clone git://git.openembedded.net/openembedded
```

OpenEmbedded è costantemente aggiornato, pertanto si consiglia di aggiornarlo frequentemente con i seguenti comandi

```
cd /home/koan/devel/openembedded
git pull
```

quindi scaricare la versione stabile dal branch **stable/2009** (vedremo in seguito i dettagli relativi a git ed i branch)

```
git checkout origin/stable/2009 -b stable/2009
git pull
```

verifichiamo quindi di essere nel branch corretto prima di procedere

```
$ git branch
  org.openembedded.dev
* stable/2009            <<<<< l'asterisco '*' ci indica dove siamo
```

5.7 Configurazione specifica di KaeilOS

Scaricare la configurazione specifica di KaeilOS. La versione qui indicata come 4.05 può risultare obsoleta al momento della pubblicazione del presente documento, si raccomanda pertanto di verificare sul sito del prodotto prima di procedere.

```
cd /home/koan/devel
wget ftp://ftp.koansoftware.com/public/kaeilos/kaeilos-4.05-config-files.tgz
tar xzvf kaeilos-4.05-config-files.tgz
```

queste operazioni creeranno un albero di directory come il seguente, che analizzeremo di seguito

```
devel/
|-- bitbake                — Subversion tree of BitBake
|-- build                  — place where I do all OE builds
|   `-- kaeilos
|       `-- conf           — all configuration files
|           |-- auto.conf
|           |-- local.conf
|           `-- site.conf
|-- openembedded           — OpenEmbedded directory
|   |-- COPYING.MIT
|   |-- MAINTAINERS
|   |-- README
|   |-- build
|   |-- classes
|   |-- conf
|   |-- contrib
|   |-- files
|   |-- recipes            - was named packages
|   |-- removal.txt
|   `-- site
|-- sources                - downloaded sources directory
`-- setup-kaeilos.sh       - environment setting script (first to be called)
```

5.8 Compilazione

La compilazione avviene secondo quanto specificato nei files di configurazione. Come primo passo è necessario impostare le variabili di environment. Ciò viene effettuato con lo script seguente:

```
cd /home/koan/devel
source setup-kaeilos.sh  /home/koan/devel/build/kaeilos
```

questo comando ci sposterà in un'altra directory (in **/home/koan/devel/build/kaeilos**) dove infine lanceremo la compilazione di un'immagine finale minimale.

L'interfaccia primaria per la compilazione di OpenEmbedded è il comando bitbake. Bitbake scaricherà e applicherà le patch necessarie attingendo dalla rete, quindi occorre sviluppare su una macchina connessa a internet. Ogni comando bitbake deve essere lanciato dentro la directory build con le variabili d'ambiente correttamente impostate.

```
bitbake console-image
```

esistono altri target di compilazione predefiniti, ad esempio per compilare un'immagine che fornisce un sistema grafico basato su X11 e GPE pronta all'uso sarà sufficiente dare il comando:

```
bitbake x11-image
```

Le immagini che possono essere generate risiedono nella directory

```
openembedded/recipes/images
```

alcune di esse sono le seguenti:

- helloworld : Compila un eseguibile d'esempio.
- minimal-image : Compila un'immagine minimale senza supporto grafico.
- x11-image : Compila un'immagine minimale senza supporto grafico.
- opie-kdepim-image : Compila un'immagine basata su OPIE e la suite KDE PIM.
- gpe-image : Compila un'immagine basata su GPE.
- virtual/kernel : Compila solo il kernel per il target configurato.

La generazione di un sistema target comporta la specifica della tripletta DISTRO/MACHINE/IMAGE come ad esempio kaeilos/ronetix-pm9263/x11-image con cui si genera un sistema grafico basato su x11 per una scheda ARM con cpu AT91SAM9263 basata sulla distribuzione KaeilOS.

5.9 Architetture supportate

Le architetture supportate sono attualmente le seguenti. Il sistema che si intende generare va specificato nel file auto.conf per mezzo del parametro MACHINE. Altrimenti è possibile specificare il parametro direttamente dalla linea comandi al momento del lancio di bitbake.

```
MACHINE ?= "at91sam9263ek"
MACHINE ?= "ronetix-pm9263"
MACHINE ?= "vortex86sx"
MACHINE ?= "at91sam9g20ek"
```

Altre architetture saranno supportate in futuro e possono anche essere aggiunte inviando le proprie patch a KOAN o alla mailing list di OpenEmbedded.

5.10 Emulazione di immagini KaeilOS con QEMU

E' possibile emulare attraverso QEMU un'immagine di KaeilOS/OE. I file immagine e il kernel sono generati all'interno della directory

```
build/tmp/deploy/glibc/images
```

Per lanciare l'emulazione è sufficiente eseguire il comando:

```
qemu -m 512 -hda x11-image-qemux86.ext2  \
       -kernel bzImage-qemux86.bin --append root=/dev/hda
```

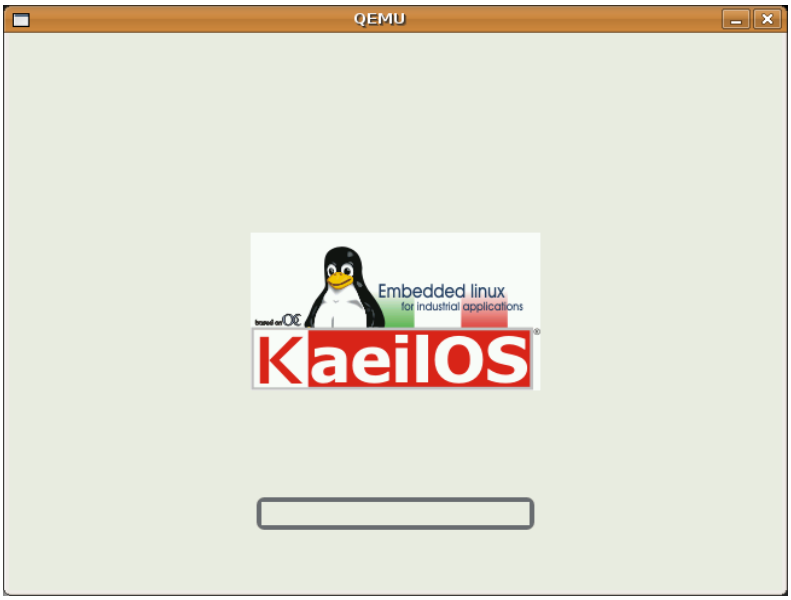

Figura 5. Boot di KaeilOS con QEMU

6 BITBAKE

Per iniziare a compilare KaeilOS/OE occorre scaricare il tool **bitbake** che ne permette la gestione. E' raccomandabile usare bitbake senza installarlo nel sistema globale ma esclusivamente nelle directory dell'ambiente di sviluppo Bitbake. Bitbake è scritto in python e non richiede compilazione per essere usato occorre esclusivamente impostare in maniera corretta la variabile PATH e rendere accessibile l'eseguibile.

6.1 Python-psyco

Per velocizzare la compilazione è utile installare Psyco, un compilatore python runtime che offre un incremento medio di velocità di esecuzione del codice pari a 4x

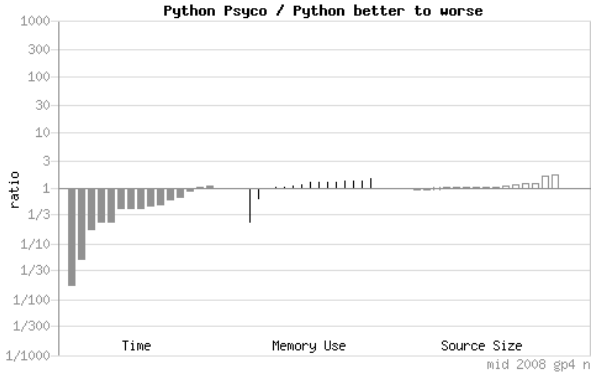

Figura 6. Confronto prestazioni tra Python-psyco e Python

http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=psyco

Purtroppo al momento Psyco presenta dei problemi con sistemi a 64 bit, pertanto è possibile utilizzarlo solo con sistemi a 32 bit.

6.2 Bitbake, opzioni

Le opzioni di **bitbake** più interessanti dal punto di vista dello sviluppo con KaeilOS/OE sono le seguenti:

--version	show program's version number and exit
-b BUILDFILE, --buildfile=BUILDFILE	execute the task against this .bb file, rather than a package from BBFILES
-c CMD, --cmd=CMD	Specify task to execute. Note that this only executes the specified task for the providee and the packages it depends on, i.e. 'compile' does not implicitly call stage for the dependencies (IOW: use only if you know what you are doing). Depending on the base.bbclass a listtaks tasks is defined and will show available tasks
-e, --environment	show the global or per-package environment
-d, --disable-psyco	disable using the psyco just-in-time compiler (not recommended)
-g, --graphviz	emit the dependency trees of the specified packages in the dot syntax

I comandi bitbake messi a disposizione da KaeilOS/OE sono:

-c clean	clean the specified package
-c fetchall	fetches all the packages
-c rebuild	clean and rebuild the specified package
--c listtasks	show the global or per-package environment
-c devshell	open a shell with all environment variables properly set

6.3 Funzionamento di bitbake

Il funzionamento di **bitbake** prevede il parsing dei files di **configurazione** generale e dei **.bb** trovati nell'albero di directory specificato. Ognuno di questi files .bb specifica una ricetta (recipe) che definisce le caratteristiche di uno specifico pacchetto. Facendo il parsing delle ricette viene creata un'area contenente i metadati del progetto.

Grazie alle informazioni così collezionate nei metadati, vengono cercate le interdipendenze tra i pacchetti e le ricette coinvolte nella compilazione. Successivamente vengono svolte le azioni specificate nei task di ogni ricetta coinvolta.

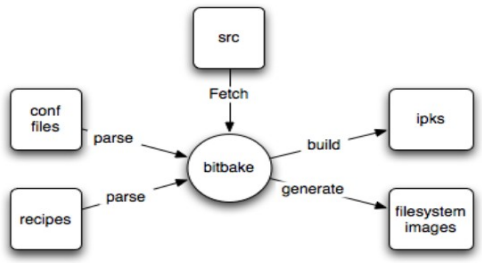

Figura 7. Schema compilazione bitbake

I pacchetti specificati, direttamente dai files di **configurazione** generale o indirettamente (a cascata) dalle ricette (.bb) da essa dipendenti contengono anche indicazioni su dove trovare il codice sorgente dei pacchetti. Il codice sorgente viene scaricato da internet, e successivamente configurato e compilato

La configurazione avviene in modo automatico con autotools per i pacchetti che dispongono dell'apposita configurazione o nel modo previsto dal pacchetto, negil altri casi.

7 GIT

Git è un sistema software di controllo versione distribuito, creato da Linus Torvalds nel 2005.

La progettazione di Git è stata ispirata da BitKeeper e da Monotone, dove ogni sviluppatore ha una copia locale dell'intera cronologia di sviluppo, e le modifiche vengono copiate da un repository a un altro. Queste modifiche vengono importate come diramazioni aggiuntive di sviluppo, e possono essere fuse allo stesso modo di una diramazione sviluppata localmente.

Git era stato pensato inizialmente solamente come motore a basso livello che altri potevano usare per scrivere un front-end. Tuttavia, il progetto Git è in seguito diventato un sistema completo di controllo versione, direttamente utilizzabile da riga di comando. Vari importanti progetti software adesso usano Git per il loro controllo versione, e principalmente il kernel di Linux.

A causa della difficoltà di utilizzo delle prime versioni, e della lunga curva di apprendimento, il programma è stato definito "il sistema di controllo versione progettato per farti sentire più stupido di quanto tu lo sia".

7.1 Git, configurazione iniziale

Sebbene non sia indispensabile, è consigliato configurare git per ottenere un'ambiente di lavoro più gradevole, con colorazione sintattica e la propria firma applicata in automatico ad eventuali patch.

La configurazione avviene a livello utente con i comandi seguenti:

```
$ git config --global user.name "Nom Cognome"
$ git config --global user.email "nome@dominio.com"
$ git config --global color.status auto
$ git config --global color.branch auto
$ git config --global color.diff auto
$ git config --global color.ui auto
$ git config --global core.editor vim
```

e si ottiene il seguente risultato nel file di configurazione

```
$ cat ~/.gitconfig
[user]
        name = Nome Cognome
        email = nome@dominio.com
[color]
        status = auto
        branch = auto
        diff = auto
```

```
        ui = auto
[core]
        editor = vim
```

7.2 Comandi git utilizzati con KaeilOS/OE

I comandi più utilizzati durante lo sviluppo con KaeilOS/OE sono quelli che permettono di gestire le patch e i branch all'interno della directory di OE. Sebbene ciò non sia un'attività normalmente necessaria per l'utente, possono tornare utili qualora si desideri implementare una propria ricetta in KaeilOS/OE o creare una propria patch.

Git pull	Fetch from and merge with another repository or a local branch
Git branch	List, create, or delete branches
Git checkout	Checkout a branch or paths to the working tree
Git add	Add file contents to the index
Git commit	Record changes to the repository
Git status	Show the working tree status
Git log	Show commit logs
Git reset	Reset current HEAD to the specified state

8 CONTRIBUTI E TRADUZIONE DI QUESTO DOCUMENTO

Qualunque contributo alla traduzione, correzione o integrazione del presente documento è gradito, e dovrà essere inviato al seguente indirizzo email: m.cavallini@koansoftware.com

Eventuali aggiornamenti del presente documento saranno pubblicati al seguente indirizzo web:

ftp://ftp.koansoftware.com/public/talks/Confsl-2009/KaeilOS-Confsl09.pdf

Bibliografia

Swicegood, Travis. (2008). Pragmatic Version Control Using Git, USA: The Pragmatic Bookshelf

Vellei, Simone. (2008). KOAN documentazione interna aziendale. Analisi frameworks per sistemi embedded

Lauer, Michael. (2005). FOSDEM presentation. Building Embedded Linux Distributions with BitBake and OpenEmbedded

Opdenacker M. & Petazzoni T. (2009). Free Electrons presentation. OpenEmbedded

Rigo, Armin. Representation-based Just-in-time Specialization and the Psyco prototype for Python

OpenEmbedded. User manual. http://docs.openembedded.org/usermanual/usermanual.pdf

Bitbake. Manual page. http://bitbake.berlios.de

Git. Manual page. http://www.kernel.org/pub/software/scm/git/docs/git.html

Wikipedia. http://it.wikipedia.org - http://en.wikipedia.org

Linux Devices.com. http://www.linuxdevices.com