Stefano Busnelli (IlGuru)

CROSS-COMPILAZIONE DISTRIBUITA PER RASPBERRY PI

10 May 2016

Per questo articolo utilizzerò due macchine che montano un sistema operativo linux di tipo debian like, nello specifico una raspbian montata su una scheda raspberry pi 3B, ed una debian 8.4 Jessie su un pc x86_64. In generale quindi occorrono:

- due computer di diverse architetture (vanno bene anche uguali, ma diventerebbe troppo facile)
- due sistemi operativi debian like (debian, raspbian, ubuntu ecc...) installati, in rete e funzionanti
- competenze minime di amministrazione di un sistema operativo linux, come installazione di un programma mancante o edit di un file in /etc

Tratteremo questi argomenti:

- toolchain di compilazione
- distcc
- ccache

Non ci occuperemo di Avahi quindi gli indirizzi IP saranno tutti hardcoded. Le informazioni contenute in questo articolo sono un riassunto di risorse come:

- man pages
- Debian Wiki
- Vari blog e risorse in rete

Le due macchine che useremo ad esempio, sono collegate in rete wifi nella stessa subnet (30.20.10.0/24) e si parlano:

- Jessie, è un normale laptop HP, architettura x86_64, processore Intel a 2.5GHz con 4 core e 4 GB di ram, IP=30.20.10.245
- Berry, è un Raspberry PI 3B, architettura armhf, processore ARMv8 a 64bit a 1.2GHz con 3 core e 1 GB di ram, IP=30.20.10.172

Cross compilatore

Appare evidente che non è possibile compilare direttamente su Jessie il codice di Berry, perchè questo non lo potrebbe eseguire. Per fare questo, dobbiamo far diventare Jessie un cross compilatore, in grado di generare file compilati per ARM, installando la tool-chain armhf. Jessie: Per prima cosa dobbiamo abilitare il repository Debian Cross-toolchains

```
# echo "deb http://emdebian.org/tools/debian/ jessie main" >> /etc/apt/sources.list.d/c

# curl http://emdebian.org/tools/debian/emdebian-toolchain-archive.key | apt-key add -
```

Ora abilitiamo l'architettura armhf

```
# dpkg --add-architecture armhf
```

Infine installiamo crossbuild-essential-armhf:

```
# apt-get update

# apt-get install crossbuild-essential-armhf
```

Poichè ora abbiamo reso visibile il repository unstable, diciamo a debian che preferiamo il repo jessie:

```
# echo "APT::Default-Release \"jessie\";" > /etc/apt/apt.conf.d/20defaultrelease
```

A questo punto eseguendo i tool in /usr/bin/ che il cui nome inizia con arm*, Jessie è in grado di compilare un file sorgente C per essere letto da una architettura che non è la sua. Creiamo un file chiamandolo hello.c fatto così:

```
#include <stdio.h>
int main() {
   printf("Hello World!\n");
   return 0;
}
```

Compiliamolo con la tool-chain nativa di Jessie:

```
# gcc -o hello.x86_64 hello.c
```

Vediamo cosa c'è dentro:

```
# file hello.x86_64

hello.x86_64: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
```

Benissimo, Jessie è ancora capace di compilare per la sua architettura.

```
# ./hello.x86_64

Hello World!
```

Ora compiliamo per ARM:

```
# /usr/bin/arm-linux-gnueabihf-gcc -o hello.arm hello.c
```

Vediamo cosa c'è dentro:

```
# file hello.arm

hello.arm: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
```

Ovviamente questo eseguibile, non può essere eseguito da Jessie, infatti:

```
# ./hello.arm

-su: ./hello.o: cannot execute binary file: Exec format error
```

Però è un eseguibile, e se Berry lo lanciasse vedrebbe la scritta Hello World! A questo punto potremmo installare un compilatore per x86_64 anche su Berry, ma non sarebbe molto utile come server di compilazione viste le ridotte capacità computazionali.

Distcc

Va bene, ora dobbiamo insegnare a Berry come farsi compilare il codice dalla performante Jessie. Istalleremo su entrambe le macchine distcc, e saranno entrambi server e client di compilazione. Nello specifico però Jessie compilerà per tutti, Berry solo per se stesso. Su Jessie e Berry:

```
# apt-get install distcc
```

Editiamo il file di configurazione di distcc che si trova in /etc/default/distcc

```
STARTDISTCC="true"

ALLOWEDNETS="127.0.0.1 30.20.10.0/24"
```

Su Jessie:

```
LISTENER="30.20.10.245
```

Su Berry:

```
LISTENER="30.20.10.172"
```

Ci sono alri due parametri che si possono configurare, che per ora ognoriamo:

```
NICE="10"
```

Con un valore da 0 a 20, dice al demone distccd quanto essere 'carino' nei confronti degli altri processi.

```
JOBS=""
```

Dice al demone quanti job alla volta accettare. Una buona idea è impostare questo valore al numero di core della macchina su cui gira il demone. Salviamo le modifiche ed avviamo i servizi: Su Jessie e Berry:

```
# service distcc restart
```

Verifichiamo che il demone sia in ascolto. Su Jessie:

```
# service distcc status
```

```
● distcc.service - LSB: simple distributed compiler server
   Loaded: loaded (/etc/init.d/distcc)
   Active: active (running) since Tue 2016-05-10 15:52:20 UTC; 9s ago
  Process: 26043 ExecStop=/etc/init.d/distcc stop (code=exited, status=0/SUCCESS)
  Process: 26049 ExecStart=/etc/init.d/distcc start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/distcc.service
           ├─26054 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           ├─26055 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           ├─26057 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           ├─26058 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           ├─26059 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           ├─26060 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log
           └─26061 /usr/bin/distccd --pid-file=/var/run/distccd.pid --log-file=/var/log

# netstat -pant | grep distcc

tcp 0 0 30.20.10.245:3632 0.0.0.0:* LISTEN 26054/distccd
```

E lo stesso può essere verificato su Berry. Ora abbiamo 2 server in ascolto ciascuno sul suo indirizzo alla porta di default 3632, ma non abbiamo ancora detto alla parte client che è quella che invia le compilazioni, dove andare a compilare. L'idea sarebbe che un client, possa dire a N server di compilazione di fare del lavoro per lui, e la lista dei server da utilizzare la prende dalla variabile d'ambiente DISTCC_HOSTS. Su Jessie:

```
# export DISTCC_HOSTS="30.20.10.245"
```

Il client di Jessie invia compilazioni solo al demone che gira su Jessie. Su Berry:

```
# export DISTCC_HOSTS="30.20.10.245 30.20.10.171"
```

Il client di Berry invia compilazioni al demone che gira su Jessie e di Berry. Verifichiamo che i job vengano distribuiti: Apriamo un terminale su Jessie e mostriamo il contenuto del log:

```
# tail -f /var/log/distccd.log
```

Lanciamo una compilazione su Berry:

```
# distcc arm-linux-gnueabihf-gcc -c hello.c

# file hello.o

hello.o: ELF 32-bit LSB relocatable, ARM, EABI5 version 1 (SYSV), not stripped
```

Nel frattempo nel log di Jessie:

```
distccd[26055] (dcc_job_summary) client: 30.20.10.172:51695 COMPILE_OK exit:0 sig:0 cor
```

Berry ha inviato il file hello.c dicendogli con quale tool-chain compilarlo, Jessie ha obbedito ed ha restituito hello.o a Berry. Jessie fa solo questo, non può linkare i binari *.o tra di loro, questa è una operazione che può fare solo Berry che lancia in giro per la rete le compilazioni e quando ha ricevuto tutti i file binari può linkarli. Per questo motivo è stata usata l'opzione -c ed il file hello.o non è eseguibile da riga di comando. Se non avessimo usato l'opzione -c distcc di Berry si sarebbe accorto che doveva eseguire una operazione che non poteva fare ed avrebbe lanciato la compilazione in locale senza chiamare in causa Jessie.

Ccache

Possiamo velocizzare ancora di più la compilazione, specialmente se sullo stesso server vengono eseguite le stesse identiche compilazioni da diversi client. L'idea è di avere una cache locale di tutto quello che viene eseguito, e quando viene richiesto di compilare un sorgente già compilato, invece di ricompilarlo viene spedito il file binario cachato. Su Jessie e Berry installiamo ccache:

```
# apt-get install ccache
```

ccache potrebbe lavorare anche da solo con il compilatore locale, ma la cosa bella è farlo lavorare assieme a distcc Diciamo a ccache quale compilatore usare impostando la variabile d'ambiente CCACHE_PREFIX. Su Jessie e Berry:

```
# export CCACHE_PREFIX="distcc"
```

Su Berry, come prima, lanciamo una compilazione distribuita:

```
# ccache arm-linux-gnueabihf-gcc -c hello.c
```

Sul log di Jessie leggiamo:

```
distccd[26057] (dcc_job_summary) client: 30.20.10.172:51697 COMPILE_OK exit:0 sig:0 cor
```

Progetti con autotools

Per utilizziare tutto l'ambaradan nei progetti con autotools, bisogna impostare 2 variabili d'ambiente: CC e CXX. Queste variabili verranno lette dai vari configure per creare i Makefile che lanceranno il compilatore che abbiamo installato in maniera opportuna. Su Berry:

```
# export CC="ccache arm-linux-gnueabihf-gcc"

# export CXX="ccache arm-linux-gnueabihf-g++"
```

Se invece avessimo voluto usare solo distcc senza ccache:

```
export CC="distcc arm-linux-gnueabihf-gcc"

export CXX="distcc arm-linux-gnueabihf-g++"
```

Proviamo a compilare screen su Berry usando ccache:

```
# apt-get build-dep screen

# apt-get source screen

# cd /screen-4.2.1

# ./configure
```

Guardiamo cosa c'è finito nel Makefile:

```
# cat Makefile | grep ccache

CC = ccache arm-linux-gnueabihf-gcc

CXX = ccache arm-linux-gnueabihf-g++
```

Lanciamo la compilazione, dicendo di pompare un pò ora che ce lo possiamo permettere. Abbiamo a disposizione 8 core, ma con più macchine potrebbero diventare anche molti di più!

```
# make -j8
```

8 job in parallelo, perchè stiamo usando il piccolo Berry in locale e la potente Jessie in remoto! Nel frattempo nel log Berry

```
distccd[15026] (dcc_job_summary) client: 30.20.10.172:54984 COMPILE_OK exit:0 sig:0 cor
distccd[15027] (dcc_job_summary) client: 30.20.10.172:54988 COMPILE_OK exit:0 sig:0 cor
distccd[15025] (dcc_job_summary) client: 30.20.10.172:54990 COMPILE_OK exit:0 sig:0 cor
```

```
distccd[15024] (dcc_job_summary) client: 30.20.10.172:54992 COMPILE_OK exit:0 sig:0 cor
distccd[15028] (dcc_job_summary) client: 30.20.10.172:54994 COMPILE_OK exit:0 sig:0 cor
distccd[15026] (dcc_job_summary) client: 30.20.10.172:54998 COMPILE_OK exit:0 sig:0 cor
distccd[15025] (dcc_job_summary) client: 30.20.10.172:55002 COMPILE_OK exit:0 sig:0 cor
distccd[15027] (dcc_job_summary) client: 30.20.10.172:55000 COMPILE_OK exit:0 sig:0 cor
distccd[15023] (dcc_job_summary) client: 30.20.10.172:54995 COMPILE_OK exit:0 sig:0 cor
distccd[15024] (dcc_job_summary) client: 30.20.10.172:55008 COMPILE_OK exit:0 sig:0 cor
distccd[15028] (dcc_job_summary) client: 30.20.10.172:55009 COMPILE_OK exit:0 sig:0 cor
distccd[15027] (dcc_job_summary) client: 30.20.10.172:55015 COMPILE_OK exit:0 sig:0 cor
distccd[15023] (dcc_job_summary) client: 30.20.10.172:55016 COMPILE_OK exit:0 sig:0 cor
distccd[15028] (dcc_job_summary) client: 30.20.10.172:55019 COMPILE_OK exit:0 sig:0 cor
distccd[15026] (dcc_job_summary) client: 30.20.10.172:55013 COMPILE_OK exit:0 sig:0 cor
distccd[15024] (dcc_job_summary) client: 30.20.10.172:55018 COMPILE_OK exit:0 sig:0 cor
distccd[15025] (dcc_job_summary) client: 30.20.10.172:55017 COMPILE_OK exit:0 sig:0 cor
```

E nel log di Jessie:

```
distccd[27815] (dcc_job_summary) client: 30.20.10.172:52221 COMPILE_OK exit:0 sig:0 cor
distccd[27788] (dcc_job_summary) client: 30.20.10.172:52223 COMPILE_OK exit:0 sig:0 cor
distccd[27808] (dcc_job_summary) client: 30.20.10.172:52224 COMPILE_OK exit:0 sig:0 cor
distccd[27819] (dcc_job_summary) client: 30.20.10.172:52227 COMPILE_OK exit:0 sig:0 cor
distccd[27782] (dcc_job_summary) client: 30.20.10.172:52225 COMPILE_OK exit:0 sig:0 cor
distccd[27768] (dcc_job_summary) client: 30.20.10.172:52229 COMPILE_OK exit:0 sig:0 cor
distccd[27815] (dcc_job_summary) client: 30.20.10.172:52231 COMPILE_OK exit:0 sig:0 cor
distccd[27808] (dcc_job_summary) client: 30.20.10.172:52235 COMPILE_OK exit:0 sig:0 cor
distccd[27788] (dcc_job_summary) client: 30.20.10.172:52233 COMPILE_OK exit:0 sig:0 cor
distccd[27782] (dcc_job_summary) client: 30.20.10.172:52239 COMPILE_OK exit:0 sig:0 cor
distccd[27768] (dcc_job_summary) client: 30.20.10.172:52241 COMPILE_OK exit:0 sig:0 cor
distccd[27819] (dcc_job_summary) client: 30.20.10.172:52237 COMPILE_OK exit:0 sig:0 cor
distccd[27808] (dcc_job_summary) client: 30.20.10.172:52243 COMPILE_OK exit:0 sig:0 cor
distccd[27815] (dcc_job_summary) client: 30.20.10.172:52242 COMPILE_OK exit:0 sig:0 cor
distccd[27788] (dcc_job_summary) client: 30.20.10.172:52244 COMPILE_OK exit:0 sig:0 cor
distccd[27782] (dcc_job_summary) client: 30.20.10.172:52245 COMPILE_OK exit:0 sig:0 cor
distccd[27768] (dcc_job_summary) client: 30.20.10.172:52250 COMPILE_OK exit:0 sig:0 cor
distccd[27819] (dcc_job_summary) client: 30.20.10.172:52252 COMPILE_OK exit:0 sig:0 cor
distccd[27808] (dcc_job_summary) client: 30.20.10.172:52248 COMPILE_OK exit:0 sig:0 cor
distccd[27815] (dcc_job_summary) client: 30.20.10.172:52249 COMPILE_OK exit:0 sig:0 cor
distccd[27788] (dcc_job_summary) client: 30.20.10.172:52258 COMPILE_OK exit:0 sig:0 cor
```

Il file compilato:

```
# file ./screen

./screen: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, i
```

Compilazione distribuita di pacchetti deb

Una volta impostate tutte le variabili d'ambiente, la cross compilazione distribuita funziona automaticamente anche nella compilazione dei sorgenti dei repository debian. Esempio su Berry:

```
# export CC="ccache arm-linux-gnueabihf-gcc"

# export CXX="ccache arm-linux-gnueabihf-g++"

# apt-get build-dep screen

# apt-get source -b screen
```

Ora apt-get oltre a scaricare i sorgenti, li compila e crea un archivio .deb che può essere installato così:

```
# dpkg -i screen-*.deb
```

Nel log file di Jessie possiamo vedere il log di compilazione di tutti i jobs.

Note

Nella macchina client, è possibile ottenere informazioni di debug impostando la variabile d'ambiente DISTCC_VERBOSE che normalmente è a 0.

```
export DISTCC_VERBOSE=1
```

Quando impostiamo le variabili d'ambiente CC e CXX, che cosa succede se come compilatori usassimo gcc e g++?

```
export CC="distcc gcc"

export CXX="distcc g++"
```

Succede che ogni macchina utilizzerà il proprio compilatore nativo, tutta la compilazione dei job avviene regolarmente, ma se i vari compilatori sono fatti per architetture diverse, alla fine i file binati non saranno linkabili tra di loro :)

Fine

Ci sono alcune cose che si possono affinare:

- Utilizzare un utente apposito con cui far girare il demone.
- Inserire le varie export nel file bashrc principale o in quello degli utenti che faranno compilazioni.
- Utilizzare dei nomi host invece che gli indirizzi ip statici.

- Utilizzare il servizio Avahi per la ricerca automatica dei server in rete.

In questo articolo l'ho fatta breve per avere da subito qualcosa di funzionante. Leggendo per bene le varie man pages le possibilità sono ancora maggiori, però per il momento tenetevela così :) Versione del documento: 1.2