machinalis

Felipe Manzano

Machinalis

16/05/2011

Temas - GCC

- Introducción
- CC
- Optimizaciones
- G++
- Funcionamiento interno GCC
- Seguridad
- Instrumentación
- Extensiones de GCC
- Crosscompilación
- Compilación distribuida
- Referencias

Compilation refers to the process of converting a program from the textual source code, in a programming language such as C or C++, into machine code, the sequence of 1's and 0's used to control the central processing unit (CPU) of the computer. This machine code is then stored in a file known as an executable file, sometimes referred to as a binary file.

de 'An Introduction to GCC'

GCC: GNU Compiler Collection

- GNU C Compiler(1987)
  - La primer versión de GCC se liberó en 1987
  - El autor original de este proyecto es Richard Stallman(el fundador de GNU)
  - Originalmente, compilaba sólamente C y sus siglas se referían a "GNU C Compiler"
- GNU Compiler Collection
  - Al 2011 vamos por la versión 4.6.x!
  - Compila C, C++, Objective-C, Objective-C++, Java, Fortran, y Ada

(Vamos a ver solo la parte de C y C++)

GCC: Características principales

- Es portable, anda en cualquier plataforma. Si en Windows también.
- Es un cross-compilador; puede compilar código para ser ejecutado en otra plataforma/arquitectura (backends)
- Se puede auto-compilar!
- Compila diferentes lenguajes (frontends)
- Otras: 'fácil' de expandir; es free(GNU)

C y C++ - disclaimer -

- C y C++ son lenguajes de bajo nivel y permiten acceder directamente a la memoria
- Hay que prestar mucha atención para no 'tocar' memoria que no se debe tocar
- Un bug de corrupción de memoria puede cambiar el comportamiento del programa por completo
- GCC tiene algunas heurísticas para detectar algunos de estos errores
- Básicamente, usa otro lenguaje

El 'Hola mundo!' en C

```
1  #include<stdio.h>
2  int main (int argc, char *argv[])
3  {
4      printf ("Hola mundo!\n");
5  }
```

- Se compila así:

  `$gcc holamundo.c -o holamundo`
- La opción `-o` le dice a dónde poner el ejecutable (por defecto: `a.out`)
- Un editor? `nano/vim/gedit/emacs`
- tabulado? `indent -i8 holamundo.c`

Prendamos todos los warnings (-Wall)

- La opción `-Wall` prende todos los warnings

```
holamundo.c: In function 'main':
holamundo.c:4: warning: control reaches end of\
                                non-void function
```

- Los mensajes que genera el GCC siempre tienen la forma:

  `file:line-number:message`

- Ver también `-Wextra` para prender aún más warnings
- Con `-Werror` las warnings son consideradas errores

Compilando desde varios archivos 1/4

- Cualquier programa medianamente serio se divide en módulos/archivos
- GCC permite compilar/componer varios archivos de código

```
1  //hello.h:
2  void hello (const char * name);
```

```
1  hello.c:
2  #include <stdio.h>
3  #include "hello.h"
4  void
5  hello (const char * name)
6  {
7      printf ("Hello, %s!\n", name);
8  }
```

Compilando desde varios archivos 2/4

```
1  //main.c:
2  #include "hello.h"
3  int
4  main (void)
5  {
6      hello ("world");
7      return 0;
8  }
```

(*) Ver `gcc -E main.c`

Compilando desde varios archivos 3/4

- `#include` vs. link, intentando compilar main

```
$ gcc -Wall main.c
$ gcc main.c
/tmp/cc4B45wZ.o: In function `main':
main.c:(.text+0x11): undefined reference to `hello'
collect2: ld returned 1 exit status
```

- Ver salida del preprocesador con:

```
$ cpp main.c
```

- O inspeccionar los objetos no definidos con "nm"

```
$ gcc main.c -c
$ nm main.o
                 U hello
        00000000 T main
```

Compilando desde varios archivos 4/4

- Se puede generar un ejecutable directamente de los archivos fuente:

```
$gcc -Wall hello.c main.c -o main
```

- Se pueden generar los archivos objeto (.o) por separado:

```
$gcc -c -Wall main.c
$gcc -c -Wall hello.c
#generando main.o y hello.o (por defecto)
```

- Y luego "linkear esos dos generando un ejecutable:

```
$gcc main.o hello.o -o main
```

- GCC se da cuenta por la extensión del archivo qué debe hacer con él (internamente usa: `cpp`, `cc`, `g++`, `ld`, `...`)

Búsqueda de cabeceras (.h)

- Las lineas `#include "FILE.h"` buscan las cabeceras en el directorio actual,
- las `#include <FILE.h>` en los configurados en el sistema
- Más caminos se pueden agregar con `-I`:
  `gcc -I /misheaders/ -Wall hello.c main.c -o main`
- y además se pueden pasar varias veces:

  `gcc -I /misheaders1/ -I /misheaders2/ code.c`
- La variable de entorno `C_INCLUDE_PATH` contiene más directorios donde buscar

Bibliotecas y signaturas por defecto (1/4)

```
1  /*printflt.c*/
2  #include<stdio.h>
3  void printflt (float f){
4      printf("%f\n",f);
5  }
6  /*printflt.h*/
7  void printflt (float f);
```

- Generamos una biblioteca libpflt.a con esta función:

```
$gcc -c printflt.c
$ar rvs libpflt.a printflt.o
```

Bibliotecas y signaturas por defecto (2/4)

```
1  /* main.c: */
2  /*#include "printflt.h"*/
3  int main (void){
4       printflt (0.12345);
5       return 0;
6  }
```

- Esto se compila así:
  `gcc -Wall main.c -lpflt -L./ -o main`

Búsqueda de bibliotecas y signaturas por defecto (3/4)

- Como no incluimos la cabecera printflt.h donde está declarada 'prinflt', GCC usa una signatura por defecto que es incompatible, en algunas arquitecturas ésto es catastrófico para cierta combinacion de tipos
- Sin embargo GCC nos avisa de esto con un warning:

```
gcc -Wall main.c -o main
main.c: In function 'main':
main.c:3: warning: implicit declaration of\
                            function 'printflt'
```

Búsqueda de bibliotecas y signaturas por defecto (4/4)

- La biblioteca con la definición de `printflt` es `libprintflt.a`
- Generalmente las bibliotecas instaladas en el sistema se encuentran en: `/lib`, y `/usr/lib`
- Para linkear con `libprintflt.a` se agrega el parámetro `-lprintflt` (GCC buscará `libprintflt.a`)
- La opción `-lNAME` intenta enlazar los archivos objeto con la biblioteca `'libNAME.a'` que debe estar en los directorios estándard
- Más directorios de búsqueda de bibliotecas de pueden agregar con `-L` o con la variable de entorno `LIBRARY_PATH`

Directorios donde buscar cabeceras y bibliotecas.

- En resumen, GCC busca las cabeceras y bibliotecas:
  - en los directorios pasados con `-I` y `-L`
  - o en las variables de entorno:

    ```
    C_INCLUDE_PATH=path1:path2:path3
    LIBRARY_PATH=path1:path2:path3
    ```

  - y por último en las preconfiguradas en gcc (ver gcc -v) (/usr/include/, /usr/lib/, /lib/)

El preprocesador - ejemplito

```
1  /* test.c: */
2  #include <stdio.h>
3  int main (void) {
4  #ifdef TEST
5      printf ("Test mode\n");
6  #endif
7      printf ("Running...\n");
8      return 0;
9  }
```

El preprocesador

- El preprocesador ejecuta antes que la compilación cambiando posiblemente el programa
- Las 'variables' de pre-procesamiento se pueden definir:
  - con `#define` desde cualquier parte del código:
    ```
    #define TEST 1
    ```
  - o desde la línea de comando con `-D` p.ej.
    ```
    gcc -DTEST=1 test.c -o test
    ```
- Para investigar el código después del pre procesamiento hacer:

  ```
  gcc -E -DTEST test.c
  gcc -E test.c
  ```

Compilar para depuración

- Esto introduce meta-datos de depuración en el ejecutable
  `gcc -ggdb -o test test.c`
- Para mantener una copia de la información de depuración
  `cp test test.debug`
  `strip --only-keep-debug test.debug`
- Para borrar esta información antes de un release se puede hacer: strip test
- Para volver a introducir la info de debug:
  `objcopy --add-gnu-debuglink=test.debug test`
- Cuidado con las optimizaciones. Ej: `-fomit-frame-pointer` y `-O4`

Optimizaciones

- Las optimizaciones se controlan principalmente con el parámetro `-Ox` (`0,1,2,3,s`)
- Otro parámetro interesante es `-funroll-loops`
- Desde el código mismo se pueden usar funciones inline y variables en registros
- También se pueden usar fragmentos de assembler...

```
int x,n;
asm ("popcnt %1, %0"
      : "=r" (n) /*outputs*/
      : "r" (x)  /*inputs*/
      /*no clobbered*/
      );
```

- `gcc -s file.c genera file.s`

Probamos esto con diferentes optimizaciones..

```
 1  #include <stdio.h>
 2  double powern (double d, unsigned n) {
 3      double x = 1.0;
 4      unsigned j;
 5      for (j = 1; j <= n; j++)
 6          x *= d;
 7      return x;
 8  }
 9  int main (void) {
10      double sum = 0.0;
11      unsigned i;
12      for (i = 1; i <= 1000000000; i++) {
13          sum += powern (i, i % 5);
14      }
15      printf ("sum = %g\n", sum);
16      return 0;
17  }
```

Comparativas de las optimizaciones

```
gcc -Os test.c -o test                          0m6.738s ?
gcc -O0 test.c -o test                          0m6.413s
gcc -O1 test.c -o test                          0m4.114s
gcc -O2 test.c -o test                          0m3.200s
gcc -O3 test.c -o test                          0m3.196s
gcc -O3  -funroll-loops test.c -o test  0m2.856s
clang(llvm2.8)                                  0m2.635s
(Intel(R) Core(TM)2 CPU          T5500  @ 1.66GHz)
```

Hola mundo en C++ (1/2)

```
1  include <iostream>
2  int main ()
3  {
4      std::cout << "Hola c++!" << std::endl;
5      return 0;
6  }
```

- Se compila así:

```
$ g++ holamundo.cc -o holamundocc
$ holamundocc
Hola c++!
```

Hola mundo en C++ (2/2)

- Generalmente se usa `.cc`, `.cpp`, `.cxx` o `.C`
- `GCC` provee el compilador de c++ 'g++'
- Ambos compiladores comparten muchos de los parámetros
- Los programas en `C++` DEBEN enlazarse usando `g++` para que se utilicen las bibliotecas apropiadas
- Las bibliotecas de C++ son parte de GCC

Templates 1/3 - libstdc++

- Los templates se consideran una especie de *super* macro
- Cuando una clase con template se usa con determinado tipo de datos, la 'plantilla' correspondiente se compila con este tipo de dato sustituido donde corresponde
- Los templates estándar como `'list'` se usan directamente

```
1  /* lists.c: */
2  #include <list>
3  #include <string>
4  #include <iostream>
5  using namespace std;
6  int main () {
7      list<string> list;
8      list.push_back("Hello");
9      list.push_back("World");
10     cout << "List size = " << list.size() << endl;
11     return 0;
12 }
13
14 //g++ -Wall string.cc
```

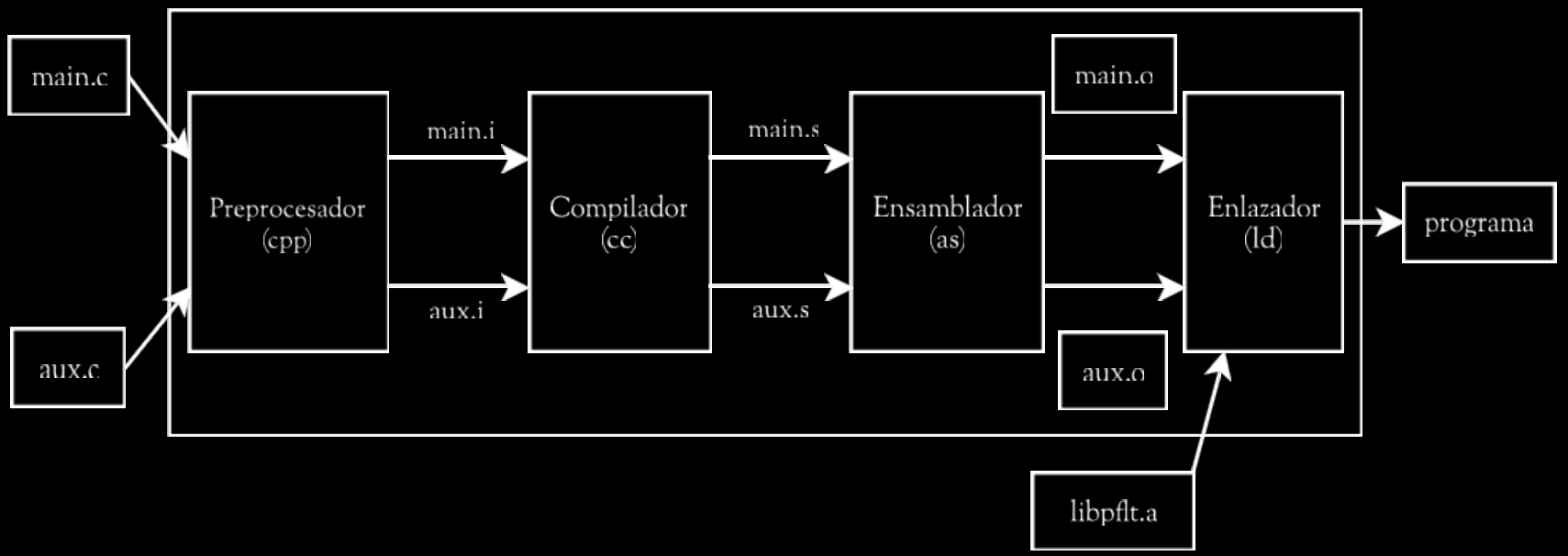
Templates 2/3 - Mis templates

- Se pueden definir templates a gusto
- Se recomienda incluir tanto la declaración como la definición de las clases con templates en la cabecera (inclusion compilation model)
- Se usan macros del pre-procesador para evitar que la clase se parsee varias veces
- `g++` compila la definición apropiada de la clase del `.h` en el momento en que es usada
- La versión compilada de la clase se pone en el archivo objeto (`.o`)
- Funciona sólo con el GNU Linker

Templates 3/3 - aha, pero no quiero poner todo en el `.h`

- Instanciación EXPLÍCITA de templates: `-fno-implicit-templates`
- Las funciones 'templatizadas' no se compilan en el punto en el que son usadas
- `g++` busca los keywords `template ClassName<type>` y fuerza la compilación de la versión del template especificada
- Si se hace bien esto asegura que la compilación de cierto template para cierto tipo aparezca sólo una vez en los archivos objeto (`.o`)
- Contra: es necesario saber de antemano qué instanciaciones se van a necesitar

Funcionamiento interno del GCC (1/2)



- GCC pasa por las siguientes etapas:
  1. Pre procesamiento: expande las macros (`gcc -E`)
  2. Compilación: de código fuente a assembler (`gcc -S`)
  3. Ensamblado: de assembler a código máquina (`as`)
  4. Enlazado: arma el ejecutable a partir de los archivos objeto (`ld`)

Funcionamiento interno del GCC (2/2)

```
1  #include <stdio.h>
2  int
3  main (void)
4  {
5      printf ("Hello, mundo!\n");
6      return 0;
7  }
```

- `cpp holamundo.c -o holamundo.i` `#(gcc -E)`
- `gcc -S holamundo.i -o holamundo.s` `#(gcc -S)`
- `as holamundo.s -o holamundo.o` `#(gcc -c)`
- `ld holamundo.o ???????? -o holamundo.exe` (Hay que incluir muchas bibliotecas estandar y de inicialización ver `gcc -v holamundo.o`)

Opciones específicas de la plataforma

- Son las opciones que empiezan con `-m`
- `-march=xxx` decide el procesador destino (`native`, `athlon`, `core2`)
- `-mtune=xxx` selecciona sólo el ordenamiento(scheduling) de instrucciones
- Cada procesador tiene su conjunto de opciones p.ej en Intel et.al. :
  - `-m32` compila para 32 bits en hosts de 64
  - `-msse4` habilita el uso de instrucciones específicas
- Ejemplo optimize.c específico para mi máquina:

```
$ gcc -march=native -O3 optimize.c -funroll-loops
                          -o optimizeGCCO3UnrollCore2
$ time ./optimizeGCCO3UnrollCore2  0m1.168s !!
```

Seguridad 1/2 - Anti stack overflow

- La opción `-fstack-protector` agrega protección contra ataques por desbordamientos de la pila (también `-fstack-protector-all`)
  - El código binario queda un poco mas grande y lento
  - A cada función pertinente se le agrega un prólogo y epílogo
  - Al principio de la función se apila un valor local implicito y 'aleatorio'
  - A la salida se revisa que este valor siga siendo el mismo que al inicio
  - Este simple check previene una importante gama de ataques
  - Y sirve para depurar más fácilmente esta clase de errores

`overflow.c:`

```
memset(alloca(1),'A',1000);
```

Seguridad 2/2 - Anti format string bug

- Se da cuando una format string es controlada por el entorno
- En Linux toda la familia de funciones de printf/syslog/scanf están afectadas
- Estos son los parámetros de GCC relacionados :
  - `-Wformat`,
  - `-Wno-format-extra-args`,
  - `-Wformat-security`,
  - `-Wformat-nonliteral`,
  - `-Wformat=2`

formatstr.c:

```
    printf("%n%n%n%n%n%n%n%n%n%n%n%n%n%n");
```

Misc 1/2 GNU Profiler (gprof)

- `gcc -pg test.c -o test` genera un binario instrumentado
- Una ejecución del ejecutable resultante genera un archivo `gmon.out`
- `gmon.out` guarda la información necesaria para contar cuánto tiempo se utilizó en cada función del programa
- Util para seleccionar qué función mejorar
- Se prueba así:

```
gcc -pg test.c -o test #binario instrumentado
./test       #genera el gmon.out silenciosamente
gprof test   #muestra los datos recopilados
```

Misc 2/2 GNU Coverage (gcov)

- Es una herramienta de testing que cuenta cuantas veces se pasa por cierta línea en una ejecución del programa
- De esta manera se podría ver qué partes del programa han sido testeadas o no
- Se usa así:

```
gcc -fprofile-arcs -ftest-coverage test.c -o test
./test     #recopila datos de la traza de ejecución
gcov test  #genera un archivo test.gcov
```

- Los archivos `.gcov` generados son una copia del código fuente con un número indicando cuántas veces se ha pasado por esa línea

Extensiones de gcc 1/3 - Inicializadores

Ref.
`http://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html`

- En ISO C99 esto:

```
1  int a[6] = { [4] = 29, [2] = 15 };
```

es lo mismo que hacer:

```
1  int a[6] = { 0, 0, 15, 0, 29, 0 };
```

- GCC permite otras macumbas:

```
1  int widths[] = { [0 ... 9] = 1,
2                   [10 ... 99] = 2,
3                   [100] = 3 };
```

Extensiones de gcc 2/3 - Arreglos de longitud zero

```
1  /* arrayzero.c: */
2  struct line {
3   int length;
4   char contents[0];
5  };
6  struct line * new_line(char* str){
7    this_length = strlen(str);
8    struct line *thisline = (struct line *)
9      malloc (sizeof (struct line) + this_length);
10   thisline->length = this_length;
11   strcpy(thisline->contents,str);
12   return thisline;
13 }
```

```
$gcc -Wall arrayzero.c -o arrayzero
```

Extensiones de gcc 3/3 - Arreglos de longitud variable

```
 1  /* bopen2.c: */
 2  int
 3  open2 (char *str1, char *str2, int flags, int mode)
 4  {
 5    char name[strlen(str1) + strlen(str2) + 1];
 6    stpcpy (stpcpy (name, str1), str2);
 7    return open (name, flags, mode);
 8  }
 9  /*
10  opcion:
11  char *name = (char *)
12           alloca (strlen(str1) + strlen(str2) + 1);
13  */
```

Crosscompilación

- Método para generar ejecutables en una arquitectura y SO (host) para ser usado en otra arquitectura y/o sistema hiperactivo (target)
- Se necesita una toolchain específica para cada combinación host/target
- Compilador que genere el assembler apropiado
- Ensamblador que genere binarios apropiados
- Linker que pueda generar ejecutables del tipo apropiados
- Bibliotecas estándares compiladas para el target específico (syscalls?)
- Existen conjuros para generar estas toolchains (ver crosstool de Dan Kegel)
- Otras toolchains para targets populares se pueden bajar directamente (android)

Crosscompilación - Hands on

> *A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run.*

- Esto se vuelve aún más interesante cuando el target es embedded
- Otro motivo: compilación distribuida

Crosscompilación - El toolchain

- Necesitamos un compilador, un ensamblador, un enlazador y las bibliotecas que ejecutarán en el destino; es decir un toolchain
- Varios scripts que automatizan la creacion de esto:
  - Dan Kegel - http://www.kegel.com/crosstool/
  - Gentoo Cross - toolchain generator - emerge crossdev
  - crosstool-ng - http://crosstool-ng.org/download/crosstool-ng

Crosscompilación - crosstool-ng

```
$ mkdir ~/crosstool-ng
$ cd ~/crosstool-ng
$ wget http://crosstool-ng.org/download/crosstool-ng/
                              crosstool-ng-1.11.1.tar.bz2
$ sha1sum  crosstool-ng-1.11.1.tar.bz2
7d992265fc77c19331da0d61b0c8fb58209f4998
```

Crosscompilación - crosstool-ng

```
$ tar jxvf crosstool-ng-1.11.1.tar.bz2
$ cd crosstool-ng-1.11.1
$#sudo zypper install make gcc bison flex texinfo libtool
$#     cvs patch lzma ncurses-devel gcc-c++ glibc-static
$ ./configure --prefix=/opt/crosstool-ng
$ make
$ sudo make install
```

Crosscompilación - Configuración de la toolchain

- Copiar `PENDRIVE/src` a `$HOME` para que no baje todos denuevo
- Preparar un directorio de trabajo

```
$ mkdir -p ~/crosstool-ng/crosstool-ng-build
$ cd ~/crosstool-ng/crosstool-ng-build
```

- Revisar los ejemplos pre-configurados y elegir uno

```
$ ls /opt/crosstool-ng/lib/ct-ng-1.11.1/samples/
$ cp /opt/crosstool-ng/lib/ct-ng-1.11.1/samples/
                    arm-unknown-linux-uclibcgnueabi/* .
$ mv crosstool.config .config
```

- Debe quedar un `.config` y su `uClibc.config` asociado en el directorio de trabajo (failsafe usar `.config` y `uClibc.config` provisto en el pendrive)
- Configurar, bajar y construir la toolchain

```
$ export PATH=/opt/crosstool-ng/bin:$PATH
$ ct-ng menuconfig
$ ct-ng build.4
```

Crosscompilación - Hola mundo para ARM

```c
/* armholamundo.c: */
int main(void){
    printf("Hola, ARM!\n");
return 0;
}
```

```
$ ~/x-tools/arm-unknown-linux-uclibcgnueabi/bin/\
arm-unknown-linux-uclibcgnueabi-gcc -static holamundo.c\
                                    -o holamundo.arm
$ adb -e push holamundo.arm /data/holamundo.arm
$ adb -e shell
# cd data
# chmod +755 holamundo.arm
# ./holamundo.arm
Hola, ARM!
```

distcc: Compilación distribuida

- distcc ditribuye la compilación de proyectos grandes en varias computadoras en red
- Soporta C y C++
- Genera los mismos resultados que una compilación local
- Fácil de instalar y a veces más rápido que compilar todo local
- Las diferentes máquinas sólo necesitan tener un compilador apropiado
- Ref. `http://code.google.com/p/distcc/`

distcc: Compilación distribuida 2/3

```
$ wget http://distcc.googlecode.com/files/\
                              distcc-3.1.tar.bz2
$ sha1sum distcc-3.1.tar.bz2
$30663e8ff94f13c0553fbfb928adba91814e1b3a
$ tar jxvf distcc-3.1.tar.bz2
$ cd distcc-3.1
$ ./configure
$ make
$ sudo make install
```

distcc: Compilación distribuida 3/3

- Otra: `sudo yum install distcc`
- Lo importante es que el server alcance el gcc apropiado
- Manualmente el server se usa así:

```
$ export PATH=~/x-tool/ ... .../bin:$PATH
$ gcc -v
$ distccd --daemon --allow 192.168.1.0/24
```

- y el cliente así:

```
$ export DISTCC_HOSTS="192.168.1.101 192.168.1.102"
$ CC=distcc
$ distcc -static holamundo.c -o holamundo
```

Cross-compilación distribuida!

```
cd ~/x-tools/ ... /bin
for file in *; do
   ln -s $file ${file#arm-unknown-linux-uclibcgnueabi-}
done
$ gcc -v
$ distccd --allow 192.168.1.0/24
```

Sacado de `http://plugapps.com/index.php5/Developers:_Cross-Compiling_with_distcc`

distcc: Compilación distribuida de joe 1/2

```
$ wget http://downloads.sourceforge.net/project/\
   joe-editor/JOE\ sources/joe-3.7/joe-3.7.tar.gz
$ sha1sum joe-3.7.tar.gz
54398578886d4a3d325aece52c308a939d31101d
$ tar xvf joe-3.7.tar.gz
$ cd joe-3.7
$ chmod +x configure
$ ./configure --prefix=/system/local \
--disable-curses --disable-termcap --disable-termidx \
CFLAGS=" -static $_XXFLAGS" ARCH=arm \
--host=arm-unknown-linux-uclibcgnueabi \
CC="arm-unknown-linux-uclibcgnueabi-gcc" \
CROSS_COMPILE="arm-unknown-linux-uclibcgnueabi-" \
sysconfdir=/system/usr/local/etc sysconfjoesubdir=/joe
```

Sacado de `http://www.curtisembedded.com/wiki/bin/view/Android/CrosstoolsAndBuilding`

distcc: Compilación distribuida de joe 2/2

```
$ export DISTCC_HOSTS="127.0.0.1 192.168.1.106"
$ export DISTCC_VERBOSE=1
$ time pump make -j8 CC=distcc
..
$ adb -e push joe /data/joe
$ adb -e shell
# cd data
# chmod 755 joe
# TERM=ansi ./joe archivo.txt
```

Referencias

- http://gcc.gnu.org/
- www.karunya.edu/linuxclub/resources/GCC.pdf
- info gcc
- http://crosstool-ng.org/hg/crosstool-ng/file/tip/docs/
- http://code.google.com/p/distcc/
- mailto:felipe.andres.manzano@gmail.com

the missing piece.