

Construir un juego de herramientas para compilación cruzada

Objetivo: Aprender a compilar un juego de herramientas de compilación cruzada para la biblioteca glibc

Luego de este laboratorio, ud podra:

- Configurar la herramienta *Crosstool-ng*
- Ejecutar *Crosstool-ng* y construir su propio compilador cruzado

Instalación de los paquetes necesarios

Instale los paquetes necesarios para este laboratorio (los laboratorios se realizaron utilizando la versión 18.04 de Ubuntu):

```
sudo apt-get install autoconf automake libtool libexpat1-dev \
  libncurses5-dev bison flex patch curl cvs texinfo git bc \
  build-essential subversion gawk python-dev gperf unzip \
  pkg-config wget help2man tree libtool-bin
```

```
sudo apt-get clean
```

Nota: tenga en cuenta que estos paquetes pueden estar instalados de algún laboratorio anterior.

Obteniendo Crosstool-ng

Vamos al directorio build-tools.

Para este laboratorio vamos a usar la versión de desarrollo de Crosstool-ng, la cual descargaremos utilizando git:

```
git clone https://github.com/crosstool-ng/crosstool-ng.git
cd crosstool-ng/
```

Vamos a utilizar la rama master dado que es la que tiene soporte para la version 7.3.0 de gcc.

Instalando Crosstool-ng

Podemos instalar Crosstool-ng de forma global en el sistema o mantenerlo local en su propio directorio. Vamos a elegir la última opción como solución. Como se documenta en docs/2\ - \ Installing\ crosstool-NG.txt, hacemos:

```
./bootstrap
./configure --prefix=${PROJECT_ROOT}/build-tools/crosstool-ng
make && make install
```

Nota: no utilizamos la opcion `--enable-local` dado que esta versión de desarrollo tiene un bug y no funciona.

Luego, es posible obtener la ayuda de Crosstool-ng ejecutando:

```
./ct-ng help
```

Configuración del juego de herramientas a producir

Una sola instalación de Crosstool-ng permite producir tantos juegos de herramientas como queramos, para diferentes arquitecturas, con diferentes bibliotecas C y diferentes versiones de varios componentes.

Crosstool-ng viene con un juego de archivos de configuracion listos para usar de varias configuraciones típicas: Crosstool-ng las llama *samples*. Se pueden listar utilizando `./ct-ng list-samples`.

Vamos a utilizar la muestra `arm-cortex_a8-linux-gnueabi`, que genera un juego de herramientas para la Beaglebone Black. La misma puede ser cargada ejecutando:

```
./ct-ng arm-cortex_a8-linux-gnueabi
```

Luego, para personalizar la configuración, podemos utilizar la interfaz `menuconfig`:

```
./ct-ng menuconfig
```

Vamos a revisar los campos, algunas opciones pueden ya estar asignadas. Nuestra configuración sera:

En Path and misc options:

- Seleccione `Debug crosstool-NG (CT_DEBUG_CT)` y dentro de esa opción seleccione también `Save intermediate steps (CT_DEBUG_CT_SAVE_STEPS)`. Esto nos va a permitir retomar cualquiera de los pasos de compilación en caso de una falla.
- Cambie `Local tarballs directory (CT_LOCAL_TARBALLS_DIR)` a `${PROJECT_ROOT}/download`. Este directorio es donde se van a descargar los distintos componentes a compilar.
- Cambie `Prefix directory (CT_PREFIX_DIR)` a `${PROJECT_ROOT}/tools/${CT_TARGET}`. Este es el directorio en donde el juego de herramientas se va a instalar. El valor de `${CT_TARGET}` lo calcula Crosstool-NG y lo traduce a la tupla del juego de herramientas.
- Deshabilite `Render the toolchain read-only (CT_PREFIX_DIR_RO)` a fin de poder agregar bibliotecas al juego de herramientas en un futuro.
- Asigne a `Number of parallel jobs (CT_PARALLEL_JOBS)` el doble de la cantidad de núcleos que posee la estación de trabajo (es posible consultar dicha cantidad ejecutando en una terminal: `cat /proc/cpuinfo | grep pocessor | wc -l`). El valor por defecto de trabajos en paralelo es cero, el cual instruye a Crosstool-NG a asignar la cantidad de procesadores que posee el host.
- Es posible cambiar `Maximum log level to see (CT_LOG_EXTRA)` a `DEBUG` para tener más detalles de que es lo que esta pasando durante la construcción en caso de que algo vaya mal. Esta opción hace que la generación sea mas lenta. Lo cambiaremos a `INFO` para información sobre el avance, pero sin demasiado detalle.

En Target options:

- En `Use specific FPU (CT_ARCH_FPU)` completar con `neon`.

- Seleccionar hardware (FPU) en Floating point (CT_ARCH_FLOAT_HW).

En Toolchain options:

- En Toolchain ID string (CT_TOOLCHAIN_PKGVERSION) escribimos Linux Embebido.
- Escriba en Tuple's alias (CT_TARGET_ALIASES) la cadena arm-linux. De esta forma, vamos a poder utilizar el compilador como arm-linux-gcc en lugar de arm-cortex_a8-linux-gnueabi-gcc, que es mucho más largo de escribir.

En C-library:

- Habilite el soporte para IPv6. Esto es por Buildroot (lo vamos a utilizar más tarde, y no acepta juegos de herramientas sin soporte para IPv6)

En Debug facilities:

- Elimine la opción duma (CT_DEBUG_duma).

En Companion libraries:

- Seleccione Check the companion libraries builds (CT_COMPLIBS_CHECK)

Explore las diferentes opciones disponibles navegando a través de los menus y consultando la ayuda para algunas opciones. No dude en consultar a su instructor por detalles disponibles en las opciones. Sin embargo, recuerde las pruebas de los laboratorios se realizaron con las configuraciones descritas anteriormente. Puede perder tiempo con problemas inesperados si personaliza la configuración del juego de herramientas.

Produciendo el juego de herramientas

Ahora ejecutamos:

```
./ct-ng build
```

y esperamos un rato!

La salida será algo similar a:

```
[INFO ] Performing some trivial sanity checks
[WARN ] Number of open files 1024 may not be sufficient to build the toolchain; increasing to 2048
[INFO ] Build started 20190521.214006
[INFO ] Building environment variables
[INFO ] =====
[INFO ] Retrieving needed toolchain components' tarballs
[INFO ] Retrieving needed toolchain components' tarballs: done in 192.52s (at 03:15)
[INFO ] =====
[INFO ] Extracting and patching toolchain components
[INFO ] Extracting and patching toolchain components: done in 48.80s (at 04:04)
[INFO ] Saving state to restart at step 'companion_tools_for_build'...
[INFO ] Saving state to restart at step 'companion_libs_for_build'...
[INFO ] =====
[INFO ] Installing ncurses for build
[INFO ] Installing ncurses for build: done in 28.56s (at 04:33)
[INFO ] Saving state to restart at step 'binutils_for_build'...
[INFO ] Saving state to restart at step 'companion_tools_for_host'...
[INFO ] Saving state to restart at step 'companion_libs_for_host'...
[INFO ] =====
```

```

[INFO ] Installing zlib for host
[INFO ] Installing zlib for host: done in 2.15s (at 04:36)
[INFO ] =====
[INFO ] Installing GMP for host
[INFO ] Installing GMP for host: done in 122.83s (at 06:39)
[INFO ] =====
[INFO ] Installing MPFR for host
[INFO ] Installing MPFR for host: done in 83.06s (at 08:02)
[INFO ] =====
[INFO ] Installing ISL for host
[INFO ] Installing ISL for host: done in 50.76s (at 08:53)
[INFO ] =====
[INFO ] Installing MPC for host
[INFO ] Installing MPC for host: done in 29.29s (at 09:22)
[INFO ] =====
[INFO ] Installing expat for host
[INFO ] Installing expat for host: done in 10.96s (at 09:33)
[INFO ] =====
[INFO ] Installing ncurses for host
[INFO ] Installing ncurses for host: done in 24.04s (at 09:57)
[INFO ] =====
[INFO ] Installing libiconv for host
[INFO ] Installing libiconv for host: done in 0.02s (at 09:57)
[INFO ] =====
[INFO ] Installing gettext for host
[INFO ] Installing gettext for host: done in 0.02s (at 09:57)
[INFO ] Saving state to restart at step 'binutils_for_host'...
[INFO ] =====
[INFO ] Installing binutils for host
[INFO ] Installing binutils for host: done in 256.77s (at 14:15)
[INFO ] Saving state to restart at step 'cc_core_pass_1'...
[INFO ] =====
[INFO ] Installing pass-1 core C gcc compiler
[INFO ] Installing pass-1 core C gcc compiler: done in 568.68s (at 23:50)
[INFO ] Saving state to restart at step 'kernel_headers'...
[INFO ] =====
[INFO ] Installing kernel headers
[INFO ] Installing kernel headers: done in 12.05s (at 24:20)
[INFO ] Saving state to restart at step 'libc_start_files'...
[INFO ] =====
[INFO ] Installing C library headers & start files
[INFO ] =====
[INFO ] Building for multilib 1/1: ''
[INFO ] Building for multilib 1/1: '': done in 13.93s (at 24:51)
[INFO ] Installing C library headers & start files: done in 14.05s (at 24:51)
[INFO ] Saving state to restart at step 'cc_core_pass_2'...
[INFO ] =====
[INFO ] Installing pass-2 core C gcc compiler
[INFO ] Installing pass-2 core C gcc compiler: done in 723.28s (at 37:12)
[INFO ] Saving state to restart at step 'libc_main'...
[INFO ] =====
[INFO ] Installing C library

```

```

[INFO ] =====
[INFO ] Building for multilib 1/1: ''
[INFO ] Building for multilib 1/1: '': done in 358.18s (at 43:28)
[INFO ] Installing C library: done in 358.29s (at 43:28)
[INFO ] Saving state to restart at step 'cc_for_build'...
[INFO ] Saving state to restart at step 'cc_for_host'...
[INFO ] =====
[INFO ] Installing final gcc compiler
[INFO ] Installing final gcc compiler: done in 853.38s (at 58:24)
[INFO ] Saving state to restart at step 'libc_post_cc'...
[INFO ] Saving state to restart at step 'companion_libs_for_target'...
[INFO ] =====
[INFO ] Installing libelf for the target
[INFO ] Installing libelf for the target: done in 5.26s (at 59:45)
[INFO ] =====
[INFO ] Installing expat for target
[INFO ] Installing expat for target: done in 9.33s (at 59:54)
[INFO ] =====
[INFO ] Installing ncurses for target
[INFO ] Installing ncurses for target: done in 39.79s (at 60:34)
[INFO ] Saving state to restart at step 'binutils_for_target'...
[INFO ] Saving state to restart at step 'debug'...
[INFO ] =====
[INFO ] Installing cross-gdb
[INFO ] Installing cross-gdb: done in 304.27s (at 66:54)
[INFO ] =====
[INFO ] Installing native gdb
[INFO ] Installing native gdb: done in 294.50s (at 71:49)
[INFO ] =====
[INFO ] Installing ltrace
[INFO ] Installing ltrace: done in 21.61s (at 72:10)
[INFO ] =====
[INFO ] Installing strace
[INFO ] Installing strace: done in 57.05s (at 73:08)
[INFO ] Saving state to restart at step 'test_suite'...
[INFO ] Saving state to restart at step 'finish'...
[INFO ] =====
[INFO ] Finalizing the toolchain's directory
[INFO ] Stripping all toolchain executables
[INFO ] Finalizing the toolchain's directory: done in 5.10s (at 74:35)
[INFO ] Build completed at 20190521.225440
[INFO ] (elapsed: 74:33.47)
[INFO ] Finishing installation (may take a few seconds)...

```

Estando dentro del directorio `${PROJECT_ROOT}`, podemos verificar la instalación con el siguiente comando:

```
tree -L 3 --charset=ascii tools/
```

que producirá una salida similar a:

```

tools/
`-- arm-cortex_a8-linux-gnueabihf
    |-- arm-cortex_a8-linux-gnueabihf

```

```

| |-- bin
| |-- debug-root
| |-- include
| |-- lib
| |-- sysroot
|-- bin
| |-- arm-cortex_a8-linux-gnueabi-hf-addr2line
| |-- arm-cortex_a8-linux-gnueabi-hf-ar
| |-- arm-cortex_a8-linux-gnueabi-hf-as
| |-- arm-cortex_a8-linux-gnueabi-hf-c++
| |-- arm-cortex_a8-linux-gnueabi-hf-cc -> arm-cortex_a8-linux-gnueabi-hf-gcc
| |-- arm-cortex_a8-linux-gnueabi-hf-c++filt
| |-- arm-cortex_a8-linux-gnueabi-hf-cpp
| |-- arm-cortex_a8-linux-gnueabi-hf-ct-ng.config
| |-- arm-cortex_a8-linux-gnueabi-hf-dwp
| |-- arm-cortex_a8-linux-gnueabi-hf-elfedit
| |-- arm-cortex_a8-linux-gnueabi-hf-g++
| |-- arm-cortex_a8-linux-gnueabi-hf-gcc
| |-- arm-cortex_a8-linux-gnueabi-hf-gcc-8.3.0
| |-- arm-cortex_a8-linux-gnueabi-hf-gcc-ar
| |-- arm-cortex_a8-linux-gnueabi-hf-gcc-nm
| |-- arm-cortex_a8-linux-gnueabi-hf-gcc-ranlib
| |-- arm-cortex_a8-linux-gnueabi-hf-gcov
| |-- arm-cortex_a8-linux-gnueabi-hf-gcov-dump
| |-- arm-cortex_a8-linux-gnueabi-hf-gcov-tool
| |-- arm-cortex_a8-linux-gnueabi-hf-gdb
| |-- arm-cortex_a8-linux-gnueabi-hf-gdb-add-index
| |-- arm-cortex_a8-linux-gnueabi-hf-gprof
| |-- arm-cortex_a8-linux-gnueabi-hf-ld
| |-- arm-cortex_a8-linux-gnueabi-hf-ld.bfd
| |-- arm-cortex_a8-linux-gnueabi-hf-ldd
| |-- arm-cortex_a8-linux-gnueabi-hf-ld.gold
| |-- arm-cortex_a8-linux-gnueabi-hf-nm
| |-- arm-cortex_a8-linux-gnueabi-hf-objcopy
| |-- arm-cortex_a8-linux-gnueabi-hf-objdump
| |-- arm-cortex_a8-linux-gnueabi-hf-populate
| |-- arm-cortex_a8-linux-gnueabi-hf-ranlib
| |-- arm-cortex_a8-linux-gnueabi-hf-readelf
| |-- arm-cortex_a8-linux-gnueabi-hf-size
| |-- arm-cortex_a8-linux-gnueabi-hf-strings
| |-- arm-cortex_a8-linux-gnueabi-hf-strip
| |-- arm-linux-addr2line -> arm-cortex_a8-linux-gnueabi-hf-addr2line
| |-- arm-linux-ar -> arm-cortex_a8-linux-gnueabi-hf-ar
| |-- arm-linux-as -> arm-cortex_a8-linux-gnueabi-hf-as
| |-- arm-linux-c++ -> arm-cortex_a8-linux-gnueabi-hf-c++
| |-- arm-linux-cc -> arm-cortex_a8-linux-gnueabi-hf-cc
| |-- arm-linux-c++filt -> arm-cortex_a8-linux-gnueabi-hf-c++filt
| |-- arm-linux-cpp -> arm-cortex_a8-linux-gnueabi-hf-cpp
| |-- arm-linux-ct-ng.config -> arm-cortex_a8-linux-gnueabi-hf-ct-ng.config
| |-- arm-linux-dwp -> arm-cortex_a8-linux-gnueabi-hf-dwp
| |-- arm-linux-elfedit -> arm-cortex_a8-linux-gnueabi-hf-elfedit
| |-- arm-linux-g++ -> arm-cortex_a8-linux-gnueabi-hf-g++

```

```

| |-- arm-linux-gcc -> arm-cortex_a8-linux-gnueabi-hf-gcc
| |-- arm-linux-gcc-8.3.0 -> arm-cortex_a8-linux-gnueabi-hf-gcc-8.3.0
| |-- arm-linux-gcc-ar -> arm-cortex_a8-linux-gnueabi-hf-gcc-ar
| |-- arm-linux-gcc-nm -> arm-cortex_a8-linux-gnueabi-hf-gcc-nm
| |-- arm-linux-gcc-ranlib -> arm-cortex_a8-linux-gnueabi-hf-gcc-ranlib
| |-- arm-linux-gcov -> arm-cortex_a8-linux-gnueabi-hf-gcov
| |-- arm-linux-gcov-dump -> arm-cortex_a8-linux-gnueabi-hf-gcov-dump
| |-- arm-linux-gcov-tool -> arm-cortex_a8-linux-gnueabi-hf-gcov-tool
| |-- arm-linux-gdb -> arm-cortex_a8-linux-gnueabi-hf-gdb
| |-- arm-linux-gdb-add-index -> arm-cortex_a8-linux-gnueabi-hf-gdb-add-index
| |-- arm-linux-gprof -> arm-cortex_a8-linux-gnueabi-hf-gprof
| |-- arm-linux-ld -> arm-cortex_a8-linux-gnueabi-hf-ld
| |-- arm-linux-ld.bfd -> arm-cortex_a8-linux-gnueabi-hf-ld.bfd
| |-- arm-linux-ldd -> arm-cortex_a8-linux-gnueabi-hf-ldd
| |-- arm-linux-ld.gold -> arm-cortex_a8-linux-gnueabi-hf-ld.gold
| |-- arm-linux-nm -> arm-cortex_a8-linux-gnueabi-hf-nm
| |-- arm-linux-objcopy -> arm-cortex_a8-linux-gnueabi-hf-objcopy
| |-- arm-linux-objdump -> arm-cortex_a8-linux-gnueabi-hf-objdump
| |-- arm-linux-populate -> arm-cortex_a8-linux-gnueabi-hf-populate
| |-- arm-linux-ranlib -> arm-cortex_a8-linux-gnueabi-hf-ranlib
| |-- arm-linux-readelf -> arm-cortex_a8-linux-gnueabi-hf-readelf
| |-- arm-linux-size -> arm-cortex_a8-linux-gnueabi-hf-size
| |-- arm-linux-strings -> arm-cortex_a8-linux-gnueabi-hf-strings
| `-- arm-linux-strip -> arm-cortex_a8-linux-gnueabi-hf-strip
|-- build.log.bz2
|-- include
|-- lib
| |-- bfd-plugins
| |-- gcc
| |-- ldscripts
| |-- libcc1.so -> libcc1.so.0.0.0
| |-- libcc1.so.0 -> libcc1.so.0.0.0
| `-- libcc1.so.0.0.0
|-- libexec
| `-- gcc
`-- share
    |-- gcc-8.3.0
    |-- gdb
    `-- licenses

```

19 directories, 74 files

Si hay algún problema en la compilación, podemos retornar al último paso exitoso ejecutando:

```
./ct-ng <step>+
```

Donde la lista de pasos se puede conocer con el comando:

```
./ct-ng list-steps
```

Problemas conocidos

Los archivos fuente no se encuentran en Internet

Es frecuente que Crosstool-ng falle debido a que no puede encontrar algunos archivos fuente en Internet, cuando dicho archivo fue movido o reemplazado por una versión mas reciente. Las nuevas versiones de Crosstool-ng vienen con URLs actualizadas, pero mientras tanto, es necesario una solución.

Si se encuentra con este problema, lo que puede hacer es buscar el archivo por su cuenta en Internet, y copiarlo al directorio `${PROJECT_ROOT}/build-tools/src`. Tenga en cuenta que los archivos fuente pueden estar comprimidos de diferentes formas (por ejemplo, terminar con `.gz` en lugar de `.bz2`), en cualquiera de los casos esta bien. Luego, todo lo que tiene que hacer es correr nuevamente `./ct-ng build`, y el mismo va a utilizar los fuentes que descargo.

Probando el juego de herramientas

Puede probar el juego de herramientas compilando el programa de ejemplo `hello.c` de forma estática en el directorio de los laboratorios con el comando `arm-cortex_a8-linux-gnueabi-gcc`.

```
arm-cortex_a8-linux-gnueabi-gcc -static hello.c -o hello
```

o utilizando el enlace `arm-linux-gcc` si escribió un alias al configurar el juego de herramientas (`CT_TARGET_ALIAS`):

```
arm-linux-gcc -static hello.c -o hello
```

hello.c:

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    printf("Hola Mundo!\n");
    return EXIT_SUCCESS;
}
```

Es posible utilizar el comando `file` sobre el binario para asegurarse que fue compilado correctamente para la arquitectura ARM.

Limpiando

Para liberar aproximadamente 7 GB de espacio en disco, realice un `./ct-ng clean` en el directorio fuente de Crosstool-NG. Esto va a remover el código fuente de diferentes componentes del juego de herramientas, como así también los archivos generados que son ahora innecesarios dado que el juego de herramientas fue instalado en `${PROJECT_ROOT}/tools`.

Utilizando Makefile

Un archivo Makefile que es útil para construir aplicaciones en el espacio de usuario:

```
# Embedded Linux Makefile modified from K. Yaghmour in 'Building embedded Linux Systems'
```



```
AS          = $(CROSS_COMPILE)as
AR          = $(CROSS_COMPILE)ar
CC          = $(CROSS_COMPILE)gcc
CPP         = $(CC) -E
LD          = $(CROSS_COMPILE)ld
NM          = $(CROSS_COMPILE)nm
OBJCOPY     = $(CROSS_COMPILE)objcopy
OBJDUMP     = $(CROSS_COMPILE)objdump
RANLIB      = $(CROSS_COMPILE)ranlib
READELF     = $(CROSS_COMPILE)readelf
SIZE        = $(CROSS_COMPILE)size
STRINGS     = $(CROSS_COMPILE)strings
STRIP       = $(CROSS_COMPILE)strip
```

```
export AS AR CC CPP LD NM OBJCOPY OBJDUMP RANLIB \
        READELF SIZE STRINGS STRIP
```

```
CFLAGS      = -O2 -Wall
HEADER_OPS  =
LDFLAGS     =
```

```
EXEC_NAME   = hello_command
INSTALL     = install
INSTALL_DIR = ${PROJECT_ROOT}/rootfs/bin
```

```
CFILES      = hello.c
HFILES      =
```

```
OBJS        = $(CFILES:%.c=%.o)
```

```
all: hello
```

```
.c.o:
        $(CC) $(CFLAGS) $(HEADER_OPS) -c $<
```

```
hello: $(OBJS) $(HFILES)
        $(CC) -o $(EXEC_NAME) $^ $(LDFLAGS)
```

```
install: hello
        test -d $(INSTALL_DIR) || $(INSTALL) -d -m 755 $(INSTALL_DIR)
        $(INSTALL) -m 755 $(EXEC_NAME) $(INSTALL_DIR)
```

```
clean:
        rm -f *.o $(EXEC_NAME)
```

```
distclean:
        rm -f *~
        rm -f *.o $(EXEC_NAME)
```