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.

Instalación de Qemu

Qemu está disponible en la mayoria de las distribuciones de Linux. Para instalar `qemu` ejecutamos:

```
sudo apt-get install qemu
```

Es posible listar las maquinas ARM soportadas por `qemu` desde la línea de comandos:

```
qemu-system-arm --machine help
```

```
akita                Sharp SL-C1000 (Akita) PDA (PXA270)
ast2500-evb          Aspeed AST2500 EVB (ARM1176)
borzoi               Sharp SL-C3100 (Borzoi) PDA (PXA270)
canon-a1100          Canon PowerShot A1100 IS
cheetah              Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie               Sharp SL-5500 (Collie) PDA (SA-1110)
connex               Gumstix Connex (PXA255)
cubieboard           cubietech cubieboard
emcraft-sf2          SmartFusion2 SOM kit from Emcraft (M2S010)
highbank             Calxeda Highbank (ECX-1000)
imx25-pdk            ARM i.MX25 PDK board (ARM926)
```



```
integratorcp          ARM Integrator/CP (ARM926EJ-S)
kzm                   ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb           Stellaris LM3S6965EVB
lm3s811evb            Stellaris LM3S811EVB
mainstone             Mainstone II (PXA27x)
midway                Calxeda Midway (ECX-2000)
mps2-an385            ARM MPS2 with AN385 FPGA image for Cortex-M3
mps2-an511            ARM MPS2 with AN511 DesignStart FPGA image for Cortex-M3
musicpal              Marvell 88w8618 / MusicPal (ARM926EJ-S)
n800                  Nokia N800 tablet aka. RX-34 (OMAP2420)
n810                  Nokia N810 tablet aka. RX-44 (OMAP2420)
netduino2             Netduino 2 Machine
none                  empty machine
nuri                  Samsung NURI board (Exynos4210)
palmetto-bmc          OpenPOWER Palmetto BMC (ARM926EJ-S)
raspi2                Raspberry Pi 2
realview-eb           ARM RealView Emulation Baseboard (ARM926EJ-S)
realview-eb-mpcore    ARM RealView Emulation Baseboard (ARM11MPCore)
realview-pb-a8        ARM RealView Platform Baseboard for Cortex-A8
realview-pbx-a9       ARM RealView Platform Baseboard Explore for Cortex-A9
romulus-bmc           OpenPOWER Romulus BMC (ARM1176)
sabrelite             Freescale i.MX6 Quad SABRE Lite Board (Cortex A9)
smdkc210              Samsung SMDKC210 board (Exynos4210)
spitz                 Sharp SL-C3000 (Spitz) PDA (PXA270)
sx1                   Siemens SX1 (OMAP310) V2
sx1-v1                Siemens SX1 (OMAP310) V1
terrier               Sharp SL-C3200 (Terrier) PDA (PXA270)
tosa                  Sharp SL-6000 (Tosa) PDA (PXA255)
verdex                Gumstix Verdex (PXA270)
versatileab           ARM Versatile/AB (ARM926EJ-S)
versatilepb           ARM Versatile/PB (ARM926EJ-S)
vexpress-a15          ARM Versatile Express for Cortex-A15
vexpress-a9           ARM Versatile Express for Cortex-A9
virt-2.10             QEMU 2.10 ARM Virtual Machine
virt                  QEMU 2.11 ARM Virtual Machine (alias of virt-2.11)
virt-2.11             QEMU 2.11 ARM Virtual Machine
virt-2.6              QEMU 2.6 ARM Virtual Machine
virt-2.7              QEMU 2.7 ARM Virtual Machine
virt-2.8              QEMU 2.8 ARM Virtual Machine
virt-2.9              QEMU 2.9 ARM Virtual Machine
xilinx-zynq-a9        Xilinx Zynq Platform Baseboard for Cortex-A9
z2                    Zipit Z2 (PXA27x)
```

Para los laboratorios vamos a utilizar la plataforma `vexpress-a9`. La plataforma Versatile Express provee de un entorno para desarrollos de diseno SoC. Posee las siguientes caracteristicas:

- V2P-CA9: 4x Cortex-A9 400MHz
- Architectura: ARMv7 Cortex-A
- Procesador: ARM Cortex-A
- RAM: 1GB
- SD: Full SD

- USB: 2
- Ethernet: Gigabit

Configuración del espacio de trabajo del proyecto

Cree el espacio de trabajo, a través de los siguientes commandos (K. Yaghmour, 'Building embedded Linux Systems'):

```
export PROJECT=dev_qemu_glibc_labs
export PROJECT_ROOT=`pwd`/${PROJECT}

mkdir -p ${PROJECT_ROOT}/{bootloader,\
build-tools/src,doc,images,kernel,\
project,rootfs,sysapps,tmp,tools}

cat > ./dev_qemu_glibc << "EOF"
export PROJECT=dev_qemu_glibc_labs
export PROJECT_ROOT=`pwd`/${PROJECT}
export ARCH=arm
export TARGET=arm-unknown-linux-gnueabihf
export CROSS_COMPILE=${TARGET}-
export PREFIX=${PROJECT_ROOT}/tools
export TARGET_PREFIX=${PREFIX}/${TARGET}
export PATH=${TARGET_PREFIX}/bin:${PATH}
cd ${PROJECT_ROOT}
EOF

. dev_qemu_glibc
```

Asegurese que los parámetros se asignaron de forma correcta!!! (`echo $PROJECT_ROOT`)

De ahora en adelante, antes de comenzar a trabajar en los laboratorios, puede iniciar las variables de entorno necesarias ejecutando el shell script `dev_qemu_glibc`.

Obteniendo Crosstool-ng

Vamos al directorio `build-tools`.

Para este laboratorio vamos a usar la versión de desarrllo 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 elejir 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-unknown-linux-gnueabi`, que genera un juego de herramientas para el modelo `vexpress-a9` de ARM. La misma puede ser cargada ejecutando:

```
./ct-ng arm-unknown-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 Architecture level (CT_ARCH_ARCH) escribimos armv7-a
- Asignamos en Tune for CPU (CT_ARCH_TUNE) la cadena cortex-a9
- En Use specific FPU (CT_ARCH_FPU) el valor vfpv3-d16 o neon.
- Seleccionamos hardware (FPU) for 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_ALIAS) la cadena arm-linux. De esta forma, vamos a poder utilzar el compilador como arm-linux-gcc en lugar de arm-unknown-linux-gnueabihf, que es mucho más largo de escribir.

En Operating System:

- Cambie Version of Linux (LINUX_V_4_14) a una version anterior a la que tenga en la estacion de trabajo (uname -a), esto se debe a que vamos a utilizar Qemu para la emulación.

En C-library:

- En glibc version seleccionar la versión 2.25.
- 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:

- Habilite las opciones gdb (CT_DEBUG_gdb), strace (DEBUG_strace) y ltrace (CT_DEBUG_ltrace). 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 descriptas 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
[INFO ]  Build started 20180903.221429
[INFO ]  Building environment variables
[INFO ]  ==================================================================
[INFO ]  Retrieving needed toolchain components' tarballs
[INFO ]  Retrieving needed toolchain components' tarballs: done in 0.06s (at 00:02)
[INFO ]  ==================================================================
```



```
[INFO ]  Extracting and patching toolchain components
[INFO ]  Extracting and patching toolchain components: done in 0.14s (at 00:03)
[INFO ]  Saving state to restart at step 'companion_tools_for_build'...
[INFO ]  ================================================================
[INFO ]  Installing libtool for build
[INFO ]  Installing libtool for build: done in 7.04s (at 00:10)
[INFO ]  Saving state to restart at step 'companion_libs_for_build'...
[INFO ]  ================================================================
[INFO ]  Installing ncurses for build
[INFO ]  Installing ncurses for build: done in 11.38s (at 00:21)
[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 GMP for host
[INFO ]  Installing GMP for host: done in 66.17s (at 01:28)
[INFO ]  ================================================================
[INFO ]  Installing MPFR for host
[INFO ]  Installing MPFR for host: done in 31.35s (at 01:59)
[INFO ]  ================================================================
[INFO ]  Installing ISL for host
[INFO ]  Installing ISL for host: done in 30.03s (at 02:29)
[INFO ]  ================================================================
[INFO ]  Installing MPC for host
[INFO ]  Installing MPC for host: done in 22.56s (at 02:52)
[INFO ]  ================================================================
[INFO ]  Installing expat for host
[INFO ]  Installing expat for host: done in 4.62s (at 02:57)
[INFO ]  ================================================================
[INFO ]  Installing ncurses for host
[INFO ]  Installing ncurses for host: done in 11.81s (at 03:08)
[INFO ]  ================================================================
[INFO ]  Installing libiconv for host
[INFO ]  Installing libiconv for host: done in 0.00s (at 03:08)
[INFO ]  ================================================================
[INFO ]  Installing gettext for host
[INFO ]  Installing gettext for host: done in 0.00s (at 03:08)
[INFO ]  Saving state to restart at step 'binutils_for_host'...
[INFO ]  ================================================================
[INFO ]  Installing binutils for host
[INFO ]  Installing binutils for host: done in 110.56s (at 05:00)
[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 204.24s (at 08:30)
[INFO ]  Saving state to restart at step 'kernel_headers'...
[INFO ]  ================================================================
[INFO ]  Installing kernel headers
[INFO ]  Installing kernel headers: done in 5.93s (at 08:48)
[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 7.31s (at 09:09)
[INFO ]  Installing C library headers & start files: done in 7.35s (at 09:09)
[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 269.67s (at 13:51)
[INFO ]  Saving state to restart at step 'libc'...
[INFO ]  ================================================================
[INFO ]  Installing C library
[INFO ]    ================================================================
[INFO ]    Building for multilib 1/1: ''
[INFO ]    Building for multilib 1/1: '': done in 167.05s (at 16:52)
[INFO ]  Installing C library: done in 167.09s (at 16:52)
[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 330.18s (at 22:54)
[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 3.40s (at 23:52)
[INFO ]  ================================================================
[INFO ]  Installing expat for target
[INFO ]  Installing expat for target: done in 4.58s (at 23:57)
[INFO ]  ================================================================
[INFO ]  Installing ncurses for target
[INFO ]  Installing ncurses for target: done in 20.53s (at 24:18)
[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 86.21s (at 26:40)
[INFO ]  ================================================================
[INFO ]  Installing native gdb
[INFO ]  Installing native gdb: done in 120.06s (at 28:40)
[INFO ]  ================================================================
[INFO ]  Installing gdbserver
[INFO ]  Installing gdbserver: done in 34.16s (at 29:14)
[INFO ]  ================================================================
[INFO ]  Installing ltrace
[INFO ]  Installing ltrace: done in 14.14s (at 29:28)
[INFO ]  ================================================================
[INFO ]  Installing strace
[INFO ]  Installing strace: done in 27.70s (at 29:56)
[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 0.55s (at 30:57)
[INFO ]  Build completed at 20180903.224526
[INFO ]  (elapsed: 30:57.21)
[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-unknown-linux-gnueabihf
    |-- arm-unknown-linux-gnueabihf
    |   |-- bin
    |   |-- debug-root
    |   |-- include
    |   |-- lib
    |   `-- sysroot
    |-- bin
    |   |-- arm-linux-addr2line -> arm-unknown-linux-gnueabihf-addr2line
    |   |-- arm-linux-ar -> arm-unknown-linux-gnueabihf-ar
    |   |-- arm-linux-as -> arm-unknown-linux-gnueabihf-as
    |   |-- arm-linux-c++ -> arm-unknown-linux-gnueabihf-c++
    |   |-- arm-linux-c++filt -> arm-unknown-linux-gnueabihf-c++filt
    |   |-- arm-linux-cc -> arm-unknown-linux-gnueabihf-cc
    |   |-- arm-linux-cpp -> arm-unknown-linux-gnueabihf-cpp
    |   |-- arm-linux-ct-ng.config -> arm-unknown-linux-gnueabihf-ct-ng.config
    |   |-- arm-linux-dwp -> arm-unknown-linux-gnueabihf-dwp
    |   |-- arm-linux-elfedit -> arm-unknown-linux-gnueabihf-elfedit
    |   |-- arm-linux-g++ -> arm-unknown-linux-gnueabihf-g++
    |   |-- arm-linux-gcc -> arm-unknown-linux-gnueabihf-gcc
    |   |-- arm-linux-gcc-6.3.1 -> arm-unknown-linux-gnueabihf-gcc-6.3.1
    |   |-- arm-linux-gcc-ar -> arm-unknown-linux-gnueabihf-gcc-ar
    |   |-- arm-linux-gcc-nm -> arm-unknown-linux-gnueabihf-gcc-nm
    |   |-- arm-linux-gcc-ranlib -> arm-unknown-linux-gnueabihf-gcc-ranlib
    |   |-- arm-linux-gcov -> arm-unknown-linux-gnueabihf-gcov
    |   |-- arm-linux-gcov-tool -> arm-unknown-linux-gnueabihf-gcov-tool
    |   |-- arm-linux-gdb -> arm-unknown-linux-gnueabihf-gdb
    |   |-- arm-linux-gprof -> arm-unknown-linux-gnueabihf-gprof
    |   |-- arm-linux-ld -> arm-unknown-linux-gnueabihf-ld
    |   |-- arm-linux-ld.bfd -> arm-unknown-linux-gnueabihf-ld.bfd
    |   |-- arm-linux-ld.gold -> arm-unknown-linux-gnueabihf-ld.gold
    |   |-- arm-linux-ldd -> arm-unknown-linux-gnueabihf-ldd
    |   |-- arm-linux-nm -> arm-unknown-linux-gnueabihf-nm
    |   |-- arm-linux-objcopy -> arm-unknown-linux-gnueabihf-objcopy
    |   |-- arm-linux-objdump -> arm-unknown-linux-gnueabihf-objdump
    |   |-- arm-linux-populate -> arm-unknown-linux-gnueabihf-populate
    |   |-- arm-linux-ranlib -> arm-unknown-linux-gnueabihf-ranlib
    |   |-- arm-linux-readelf -> arm-unknown-linux-gnueabihf-readelf
    |   |-- arm-linux-size -> arm-unknown-linux-gnueabihf-size
    |   |-- arm-linux-strings -> arm-unknown-linux-gnueabihf-strings
```



```
|   |-- arm-linux-strip -> arm-unknown-linux-gnueabihf-strip
|   |-- arm-unknown-linux-gnueabihf-addr2line
|   |-- arm-unknown-linux-gnueabihf-ar
|   |-- arm-unknown-linux-gnueabihf-as
|   |-- arm-unknown-linux-gnueabihf-c++
|   |-- arm-unknown-linux-gnueabihf-c++filt
|   |-- arm-unknown-linux-gnueabihf-cc -> arm-unknown-linux-gnueabihf-gcc
|   |-- arm-unknown-linux-gnueabihf-cpp
|   |-- arm-unknown-linux-gnueabihf-ct-ng.config
|   |-- arm-unknown-linux-gnueabihf-dwp
|   |-- arm-unknown-linux-gnueabihf-elfedit
|   |-- arm-unknown-linux-gnueabihf-g++
|   |-- arm-unknown-linux-gnueabihf-gcc
|   |-- arm-unknown-linux-gnueabihf-gcc-6.3.1
|   |-- arm-unknown-linux-gnueabihf-gcc-ar
|   |-- arm-unknown-linux-gnueabihf-gcc-nm
|   |-- arm-unknown-linux-gnueabihf-gcc-ranlib
|   |-- arm-unknown-linux-gnueabihf-gcov
|   |-- arm-unknown-linux-gnueabihf-gcov-tool
|   |-- arm-unknown-linux-gnueabihf-gdb
|   |-- arm-unknown-linux-gnueabihf-gprof
|   |-- arm-unknown-linux-gnueabihf-ld
|   |-- arm-unknown-linux-gnueabihf-ld.bfd
|   |-- arm-unknown-linux-gnueabihf-ld.gold
|   |-- arm-unknown-linux-gnueabihf-ldd
|   |-- arm-unknown-linux-gnueabihf-nm
|   |-- arm-unknown-linux-gnueabihf-objcopy
|   |-- arm-unknown-linux-gnueabihf-objdump
|   |-- arm-unknown-linux-gnueabihf-populate
|   |-- arm-unknown-linux-gnueabihf-ranlib
|   |-- arm-unknown-linux-gnueabihf-readelf
|   |-- arm-unknown-linux-gnueabihf-size
|   |-- arm-unknown-linux-gnueabihf-strings
|   `-- arm-unknown-linux-gnueabihf-strip
|-- build.log.bz2
|-- include
|   `-- gdb
|-- lib
|   |-- 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-6.3.1
    |-- gdb
    |-- info
    `-- man

20 directories, 70 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-unknown-linux-gnueabihf-gcc`.

```
arm-unknown-linux-gnueabihf-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.

Luego podemos utilizar `qemu` para probar el ejecutable:

```
qemu-arm hello
```


Limpiando

Para liberar aproximadamente 7 GB de espacio en disco, realize 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 asi tambien los archivos generados que son ahora inecesarios dado que el juego de herramientas fue instalado en `${PROJECT_ROOT}/tools`.

Utilizando Makefile

Un archivo `Makefile` que es util 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)
```

