Construir un juego de herramientas de compilación cruzada

Objetivo: Aprender a compilar un juego de herramientas de compilación cruzada con la biblioteca glibc para VExpress-a9

Luego de este laboratorio, ud podra:

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

Prerequisitos:

- Laboratorio "Configuración del entorno para Qemu"

Instalación de los paquetes necesarios

Instale los paquetes necesarios para este laboratorio:

```
sudo apt-get install autoconf 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 GNU/Linux. Para instalar `qemu` ejecutamos:

```
sudo apt-get install qemu-system-arm qemu-user
```

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

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

Supported machines are:
akita                Sharp SL-C1000 (Akita) PDA (PXA270)
ast2500-evb          Aspeed AST2500 EVB (ARM1176)
ast2600-evb          Aspeed AST2600 EVB (Cortex A7)
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)
```

© 2021 UNER, CC BY-SA license

```
cubieboard           cubietech cubieboard (Cortex-A8)
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)
mcimx6ul-evk         Freescale i.MX6UL Evaluation Kit (Cortex A7)
mcimx7d-sabre        Freescale i.MX7 DUAL SABRE (Cortex A7)
microbit             BBC micro:bit
midway               Calxeda Midway (ECX-2000)
mps2-an385           ARM MPS2 with AN385 FPGA image for Cortex-M3
mps2-an505           ARM MPS2 with AN505 FPGA image for Cortex-M33
mps2-an511           ARM MPS2 with AN511 DesignStart FPGA image for Cortex-M3
mps2-an521           ARM MPS2 with AN521 FPGA image for dual Cortex-M33
musca-a              ARM Musca-A board (dual Cortex-M33)
musca-b1             ARM Musca-B1 board (dual Cortex-M33)
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)
swift-bmc            OpenPOWER Swift BMC (ARM1176)
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-2.11            QEMU 2.11 ARM Virtual Machine
virt-2.12            QEMU 2.12 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
```

 © 2021 UNER, CC BY-SA license

```
virt-3.0            QEMU 3.0 ARM Virtual Machine
virt-3.1            QEMU 3.1 ARM Virtual Machine
virt-4.0            QEMU 4.0 ARM Virtual Machine
virt-4.1            QEMU 4.1 ARM Virtual Machine
virt                QEMU 4.2 ARM Virtual Machine (alias of virt-4.2)
virt-4.2            QEMU 4.2 ARM Virtual Machine
witherspoon-bmc     OpenPOWER Witherspoon BMC (ARM1176)
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
- Flash: 128MB
- SD: Full SD
- USB: 2
- Ethernet: Gigabit

Documentación: https://developer.arm.com/documentation/dui0448/i/hardware-description

Obteniendo Crosstool-ng

Vamos al directorio `build-tools`.

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

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

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, hacemos:

```
./bootstrap
./configure --prefix=`pwd`
make && make install
```

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

© 2021 UNER, CC BY-SA license

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 processor | 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` que por defecto está en `EXTRA (CT_LOG_EXTRA)` a `DEBUG (CT_LOG_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 (CT_LOG_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`.
- En `Tuple's vendor string (CT_TARGET_VENDOR)` escribimos `qemu`.

 © 2021 UNER, CC BY-SA license

- 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-qemu-linux-gnueabihf-gcc`, que es mucho más largo de escribir.

En `Operating System`:

- Cambie `Version of Linux (LINUX_V_5_13)` 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 `extra config (CT_GLIBC_EXTRA_CONFIG_ARRAY)` agregue `--enable-obsolete-rpc`. Esta opción fuerza generar el juego de herramientas con una cabecera obsoleta (rpc.h), la cual es utilizada por distintos paquetes (esto es solo un problema si estamos utilizando glibc).

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 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
[WARN ]  Number of open files 1024 may not be sufficient to build the toolchain;
    increasing to 2048
[INFO ]  Build started 20210824.023019
[INFO ]  Building environment variables
[INFO ]  ================================================================
[INFO ]  Retrieving needed toolchain components' tarballs
[INFO ]  Retrieving needed toolchain components' tarballs: done in 0.48s (at 00:02)
[INFO ]  ================================================================
[INFO ]  Extracting and patching toolchain components
[INFO ]  Extracting and patching toolchain components: done in 0.52s (at 00:03)
[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 20.64s (at 00:24)
[INFO ]  Saving state to restart at step 'binutils_for_build'...
[INFO ]  Saving state to restart at step 'companion_tools_for_host'...
```

© 2021 UNER, CC BY-SA license

```
[INFO ]  Saving state to restart at step 'companion_libs_for_host'...
[INFO ]  ==================================================================
[INFO ]  Installing zlib for host
[INFO ]  Installing zlib for host: done in 1.20s (at 00:25)
[INFO ]  ==================================================================
[INFO ]  Installing GMP for host
[INFO ]  Installing GMP for host: done in 57.67s (at 01:23)
[INFO ]  ==================================================================
[INFO ]  Installing MPFR for host
[INFO ]  Installing MPFR for host: done in 39.85s (at 02:03)
[INFO ]  ==================================================================
[INFO ]  Installing ISL for host
[INFO ]  Installing ISL for host: done in 35.79s (at 02:39)
[INFO ]  ==================================================================
[INFO ]  Installing MPC for host
[INFO ]  Installing MPC for host: done in 15.22s (at 02:54)
[INFO ]  ==================================================================
[INFO ]  Installing expat for host
[INFO ]  Installing expat for host: done in 6.96s (at 03:01)
[INFO ]  ==================================================================
[INFO ]  Installing ncurses for host
[INFO ]  Installing ncurses for host: done in 20.09s (at 03:21)
[INFO ]  ==================================================================
[INFO ]  Installing libiconv for host
[INFO ]  Installing libiconv for host: done in 0.00s (at 03:21)
[INFO ]  ==================================================================
[INFO ]  Installing gettext for host
[INFO ]  Installing gettext for host: done in 0.00s (at 03:21)
[INFO ]  Saving state to restart at step 'binutils_for_host'...
[INFO ]  ==================================================================
[INFO ]  Installing binutils for host
[INFO ]  Installing binutils for host: done in 158.60s (at 06:00)
[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 444.11s (at 13:31)
[INFO ]  Saving state to restart at step 'kernel_headers'...
[INFO ]  ==================================================================
[INFO ]  Installing kernel headers
[INFO ]  Installing kernel headers: done in 12.34s (at 14:12)
[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 12.04s (at 14:51)
[INFO ]  Installing C library headers & start files: done in 12.10s (at 14:51)
[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 503.07s (at 23:41)
[INFO ]  Saving state to restart at step 'libc_main'...
```

 © 2021 UNER, CC BY-SA license

```
[INFO ]  ================================================================
[INFO ]  Installing C library
[INFO ]    ============================================================
[INFO ]    Building for multilib 1/1: ''
[INFO ]    Building for multilib 1/1: '': done in 198.08s (at 27:26)
[INFO ]  Installing C library: done in 198.13s (at 27:26)
[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 598.14s (at 38:22)
[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 3.58s (at 40:20)
[INFO ]  ================================================================
[INFO ]  Installing expat for target
[INFO ]  Installing expat for target: done in 6.01s (at 40:26)
[INFO ]  ================================================================
[INFO ]  Installing ncurses for target
[INFO ]  Installing ncurses for target: done in 24.55s (at 40:51)
[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 161.29s (at 45:27)
[INFO ]  ================================================================
[INFO ]  Installing native gdb
[INFO ]  Installing native gdb: done in 174.18s (at 48:21)
[INFO ]  ================================================================
[INFO ]  Installing ltrace
[INFO ]  Installing ltrace: done in 15.27s (at 48:36)
[INFO ]  ================================================================
[INFO ]  Installing strace
[INFO ]  Installing strace: done in 49.84s (at 49:26)
[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 5.02s (at 51:33)
[INFO ]  Build completed at 20210824.032151
[INFO ]  (elapsed: 51:32.19)
[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:

© 2021 UNER, CC BY-SA license

```
tools/
`-- arm-qemu-linux-gnueabihf
    |-- arm-qemu-linux-gnueabihf
    |   |-- bin
    |   |-- debug-root
    |   |-- include
    |   |-- lib
    |   `-- sysroot
    |-- bin
    |   |-- arm-linux-addr2line -> arm-qemu-linux-gnueabihf-addr2line
    |   |-- arm-linux-ar -> arm-qemu-linux-gnueabihf-ar
    |   |-- arm-linux-as -> arm-qemu-linux-gnueabihf-as
    |   |-- arm-linux-c++ -> arm-qemu-linux-gnueabihf-c++
    |   |-- arm-linux-c++filt -> arm-qemu-linux-gnueabihf-c++filt
    |   |-- arm-linux-cc -> arm-qemu-linux-gnueabihf-cc
    |   |-- arm-linux-cpp -> arm-qemu-linux-gnueabihf-cpp
    |   |-- arm-linux-ct-ng.config -> arm-qemu-linux-gnueabihf-ct-ng.config
    |   |-- arm-linux-dwp -> arm-qemu-linux-gnueabihf-dwp
    |   |-- arm-linux-elfedit -> arm-qemu-linux-gnueabihf-elfedit
    |   |-- arm-linux-g++ -> arm-qemu-linux-gnueabihf-g++
    |   |-- arm-linux-gcc -> arm-qemu-linux-gnueabihf-gcc
    |   |-- arm-linux-gcc-11.2.0 -> arm-qemu-linux-gnueabihf-gcc-11.2.0
    |   |-- arm-linux-gcc-ar -> arm-qemu-linux-gnueabihf-gcc-ar
    |   |-- arm-linux-gcc-nm -> arm-qemu-linux-gnueabihf-gcc-nm
    |   |-- arm-linux-gcc-ranlib -> arm-qemu-linux-gnueabihf-gcc-ranlib
    |   |-- arm-linux-gcov -> arm-qemu-linux-gnueabihf-gcov
    |   |-- arm-linux-gcov-dump -> arm-qemu-linux-gnueabihf-gcov-dump
    |   |-- arm-linux-gcov-tool -> arm-qemu-linux-gnueabihf-gcov-tool
    |   |-- arm-linux-gdb -> arm-qemu-linux-gnueabihf-gdb
    |   |-- arm-linux-gdb-add-index -> arm-qemu-linux-gnueabihf-gdb-add-index
    |   |-- arm-linux-gprof -> arm-qemu-linux-gnueabihf-gprof
    |   |-- arm-linux-ld -> arm-qemu-linux-gnueabihf-ld
    |   |-- arm-linux-ld.bfd -> arm-qemu-linux-gnueabihf-ld.bfd
    |   |-- arm-linux-ld.gold -> arm-qemu-linux-gnueabihf-ld.gold
    |   |-- arm-linux-ldd -> arm-qemu-linux-gnueabihf-ldd
    |   |-- arm-linux-lto-dump -> arm-qemu-linux-gnueabihf-lto-dump
    |   |-- arm-linux-nm -> arm-qemu-linux-gnueabihf-nm
    |   |-- arm-linux-objcopy -> arm-qemu-linux-gnueabihf-objcopy
    |   |-- arm-linux-objdump -> arm-qemu-linux-gnueabihf-objdump
    |   |-- arm-linux-populate -> arm-qemu-linux-gnueabihf-populate
    |   |-- arm-linux-ranlib -> arm-qemu-linux-gnueabihf-ranlib
    |   |-- arm-linux-readelf -> arm-qemu-linux-gnueabihf-readelf
    |   |-- arm-linux-size -> arm-qemu-linux-gnueabihf-size
    |   |-- arm-linux-strings -> arm-qemu-linux-gnueabihf-strings
    |   |-- arm-linux-strip -> arm-qemu-linux-gnueabihf-strip
    |   |-- arm-qemu-linux-gnueabihf-addr2line
    |   |-- arm-qemu-linux-gnueabihf-ar
    |   |-- arm-qemu-linux-gnueabihf-as
    |   |-- arm-qemu-linux-gnueabihf-c++
    |   |-- arm-qemu-linux-gnueabihf-c++filt
    |   |-- arm-qemu-linux-gnueabihf-cc -> arm-qemu-linux-gnueabihf-gcc
    |   |-- arm-qemu-linux-gnueabihf-cpp
```

 © 2021 UNER, CC BY-SA license

```
|   |-- arm-qemu-linux-gnueabihf-ct-ng.config
|   |-- arm-qemu-linux-gnueabihf-dwp
|   |-- arm-qemu-linux-gnueabihf-elfedit
|   |-- arm-qemu-linux-gnueabihf-g++
|   |-- arm-qemu-linux-gnueabihf-gcc
|   |-- arm-qemu-linux-gnueabihf-gcc-11.2.0
|   |-- arm-qemu-linux-gnueabihf-gcc-ar
|   |-- arm-qemu-linux-gnueabihf-gcc-nm
|   |-- arm-qemu-linux-gnueabihf-gcc-ranlib
|   |-- arm-qemu-linux-gnueabihf-gcov
|   |-- arm-qemu-linux-gnueabihf-gcov-dump
|   |-- arm-qemu-linux-gnueabihf-gcov-tool
|   |-- arm-qemu-linux-gnueabihf-gdb
|   |-- arm-qemu-linux-gnueabihf-gdb-add-index
|   |-- arm-qemu-linux-gnueabihf-gprof
|   |-- arm-qemu-linux-gnueabihf-ld
|   |-- arm-qemu-linux-gnueabihf-ld.bfd
|   |-- arm-qemu-linux-gnueabihf-ld.gold
|   |-- arm-qemu-linux-gnueabihf-ldd
|   |-- arm-qemu-linux-gnueabihf-lto-dump
|   |-- arm-qemu-linux-gnueabihf-nm
|   |-- arm-qemu-linux-gnueabihf-objcopy
|   |-- arm-qemu-linux-gnueabihf-objdump
|   |-- arm-qemu-linux-gnueabihf-populate
|   |-- arm-qemu-linux-gnueabihf-ranlib
|   |-- arm-qemu-linux-gnueabihf-readelf
|   |-- arm-qemu-linux-gnueabihf-size
|   |-- arm-qemu-linux-gnueabihf-strings
|   `-- arm-qemu-linux-gnueabihf-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-11.2.0
    |-- gdb
    |-- info
    |-- licenses
    `-- man

21 directories, 76 files
```

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

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

© 2021 UNER, CC BY-SA license

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

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

Por ejemplo, si falla en el paso 'Installing pass-2 core C gcc compiler', podemos reestablecer desde el estado anterior con el siguiente comando:

```
./ct-ng cc_core_pass_2+
```

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}/download`. 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.

Error: "Don't set LD_LIBRARY_PATH. It screws up the build"

Si nos aparece el siguiente error:

```
[INFO ]  Performing some trivial sanity checks
[ERROR]  Don't set LD_LIBRARY_PATH. It screws up the build.
[ERROR]
...
[ERROR]
[ERROR]  (elapsed: 26365905:15.92)
[00:00] / ct-ng:261: recipe for target 'build' failed
make: *** [build] Error 1
```

es porque tenemos asignada la variable `LD_LIBRARY_PATH`, la cual tiene conflictos con Crosstool-ng.

Para solucionar el problema debemos limpiar la variable de entorno:

```
unset LD_LIBRARY_PATH
```

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-qemu-linux-gnueabihf-gcc`.

```
arm-qemu-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
```

 © 2021 UNER, CC BY-SA license

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 según K. Yaghmour ('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
```

© 2021 UNER, CC BY-SA license

```
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)
```

 © 2021 UNER, CC BY-SA license