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.

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_bbb_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_bbb_glibc << "EOF"
export PROJECT=dev_bbb_glibc_labs
export PROJECT_ROOT=`pwd`/${PROJECT}
export ARCH=arm
export TARGET=arm-cortex_a8-linux-gnueabihf
export CROSS_COMPILE=${TARGET}-
export PREFIX=${PROJECT_ROOT}/tools
export TARGET_PREFIX=${PREFIX}/${TARGET}
export PATH=${TARGET_PREFIX}/bin:${PATH}
cd ${PROJECT_ROOT}
```



```
EOF

. dev_bbb_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_bbb_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-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 `${PROJECT_ROOT}/build-tools/src`. Este directorio es donde se van a descargar los distintos componentes a compilar.
- Cambie `Prefix directory (CT_PREFIX_DIR)` a `${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_ALIAS)` la cadena `arm-linux`. De esta forma, vamos a poder utilzar el compilador como `arm-linux-gcc` en lugar de `arm-cortex_a8-linux-gnueabihf-gcc`, que es mucho más largo de escribir.

En `C-library`:

- En `extra config (CT_GLIBC_EXTRA_CONFIG_ARRAY)` agregue `--enable-obsolete-rpc`. Esta opción fuerza generar el juego de herraminetas 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`:

- Habilite la opción `gdb (CT_DEBUG_gdb)`. Elimine las opciones `strace (DEBUG_strace)`, `ltrace (CT_DEBUG_ltrace)` y `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:

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:

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 `esttica` en el directorio de los laboratorios con el comando `arm-cortex_a8-linux-gnueabihf-gcc`.

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

o utilizando `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 5 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 Syst

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)
```

