Práctica I

Compilador cruzado CrossTools-NG

Implementación de Sistemas Operativos II

Maestría en Sistemas Embebidos

Año 2022

Autor

Mg. Ing. Gonzalo Sanchez

Tabla de contenido

Registro de cambios

Revisión	Cambios realizados	Fecha
1.0	Creación del documento	23/07/2021
1.1	Cambio de título de carátula	27/07/21
1.2	Corrección en comandos de ct-ng	01/09/21
1.2.1	Actualización de fechas	27/08/22
1.2.2	Corrección de Toolchain ID String a MSE 7ma Cohorte	30/08/22

Introducción y consejos para la materia

Instalación de paquetes extra

No dude en instalar otros paquetes que pueda necesitar para su entorno de desarrollo. En particular, se recomienda la instalación de su editor de texto favorito y configurarlo a su gusto. Los editores de texto favoritos de los desarrolladores de Linux embebidos son, por supuesto, Vim y Emacs, pero también hay muchas otras posibilidades, como Visual Studio Code, GEdit, Qt Creator, CodeBlocks, Geany, Atom, etc.

Vale la pena mencionar que, por defecto, Ubuntu viene con una versión muy limitada del editor *vi*. Entonces, si desea usar *vi*, le recomendamos que use la versión con más funciones instalando el paquete *vim*, conocido usualmente como *vi mejorado*.

Algunos consejos

Los siguientes puntos pueden ser útiles a lo largo de todas las prácticas:

- Lea atentamente las instrucciones y los consejos. Mucha gente comete errores o pierde el tiempo porque se perdió una explicación o una guía.
- Lea siempre los mensajes de error con atención, en particular el primero que se emita. Algunas personas se topan con errores muy simples solo porque especificaron una ruta de archivo incorrecta y no prestaron suficiente atención al mensaje de error correspondiente. Otros errores comunes están asociados a la falta de dependencias (paquetes faltantes) que simplemente se solucionan instalandolos mediante **sudo apt install**.
- Nunca te quedes atascado con un problema extraño más de 5 minutos. Muestra tu problema a tus colegas o al instructor.
- Solo debe usar el usuario **root** para operaciones que requieran privilegios de superusuario, como ser montar un sistema de archivos, cargar un módulo del kernel, cambiar la propiedad del archivo, configurar la red. La mayoría de las tareas habituales (como descargar, extraer fuentes, compilar, etc) se pueden realizar como un usuario regular.

- Si ejecutó comandos desde un shell en modo **root** por error, es posible que su usuario ya no tenga los permisos para los archivos generados. En este caso, use el comando **chown -R** para que los nuevos archivos sean asignados nuevamente a su usuario (cambio de pertenencia).
  Ejemplo: **$ chown -R myuser.myuser linux/**

Construcción de Toolchain para trabajar - Compilación cruzada

Para poder comenzar a construir nuestro sistema, primero se debe generar una herramienta de compilación cruzada, o lo que comúnmente se conoce como *Toolchain*, porque en realidad son varias herramientas necesarias, que forman una especie de "cadena". Este Toolchain es del tipo *Cross-Compile* por el simple hecho de que se compila en una estación mucho más poderosa que la SBC, para que la compilación no insuma tanto tiempo, y corriendo los ejecutables en la SBC. De esta manera se compila en una arquitectura para correr los binarios en otra distinta.

Obtención de fuentes, configuración e instalación

El paso inicial es la creación de un directorio raíz para trabajar, que en este caso será **ISO_II**. Dentro de este directorio raíz se crea uno para cada herramienta o parte del sistema. En esta práctica dentro del directorio **ISO_II** se debe crear un directorio llamado **toolchain**. Luego de crearlo, se debe ingresar al mismo.

```
$ mkdir -p -v $HOME/ISO_II/toolchain

$ cd $HOME/ISO_II/toolchain
```

Una vez dentro del directorio de trabajo, es necesario clonar las fuentes de **Crosstool-NG**.

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

Habiendo clonado el repositorio, se debe ingresar al directorio y luego hacer un checkout a la última versión estable, entendiendo que la rama **master** es la que se utiliza para hacer el desarrollo.

NOTA: *Al momento de dictar la materia, el esquema de versionados de crosstool-NG cambió, siendo que antes (versión 1.22 y anteriores) se utilizaban ramas para las versiones estables, pero actualmente se utilizan tags.*

```
$ cd crosstool-ng

$ git checkout crosstool-ng-1.25.0
```

En este punto, trabajando con la última versión estable, se deben ejecutar dos comandos previos a la compilación de la herramienta (esto aún no es el toolchain, sino la herramienta para generarlo). Luego se procede a la compilación.

```
$ ./bootstrap

$ ./configure --enable-local

$ make
```

El comando *bootstrap* ejecuta un script para preparar la compilación de la herramienta, mientras que el script *configure* genera el archivo de configuración para la compilación a través de *make*. Note la opción *--enable-local*, la cual hace que no sea necesaria la instalación de la herramienta en el sistema.

NOTA: *Es muy posible que la ejecución de ./configure falle, dadas dependencias incumplidas. Observe los errores e instale las dependencias necesarias para la finalización de este script. Cabe la posibilidad de que el comando make también falle por dependencias que no se cumplen, aun cuando configure termine con éxito. Instale por medio de apt install los paquetes que hagan falta.*

Una vez finalizado el proceso de compilación, si el mismo fue correcto la ejecución del comando *help* debe generar una salida acorde.

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

Una sola instalación de Crosstool-ng permite producir tantos juegos de herramientas como se desee, para diferentes arquitecturas, con diferentes bibliotecas C y diferentes versiones de los componentes que componen el toolchain.

Crosstool-ng está dotado con un juego de archivos de configuración ya preparados para utilizar en distintas configuraciones típicas. Reciben el nombre de *samples*. Todas las *samples* se pueden listar utilizando el comando *list-samples*.

```
$ ./ct-ng list-samples
```

Las samples permiten que se genere un archivo de configuración mínimo para la generación del toolchain, el cual puede luego ser modificado. Para generar un archivo de configuración a partir de una muestra se debe ejecutar el script cg-nt segundo del nombre de la muestra seleccionada. Para esta materia se utilizará la muestra (*sample*) **arm-unknown-linux-gnueabi**.

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

El archivo de configuración inicial para compilar el toolchain ya generado a partir de la muestra puede ser modificado mediante el comando *menuconfig*.

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

La configuración por defecto es, para esta materia, suficiente casi en su totalidad. Solo se modificarán algunos aspectos menores. A continuación se listaran las opciones que deben ser actualizadas:

1. Seleccionando la opción **Path and misc options**:
    a. Seleccionar **Debug crosstool-NG** y especificar dentro de la misma **Save intermediate steps**. Cada paso exitoso supondrá un hito del cual puede comenzar una nueva compilación en caso de interrupción o falla de la actual.

b. En la opción **Local tarballs directory**, actualizar el valor a *$HOME/ISO_II/toolchain/src*. Este es el directorio donde se descargan los archivos tarball necesarios para la compilación del toolchain. **NOTA:** El directorio debe existir y tener los permisos correspondientes, el script no lo crea en caso de que no exista.
c. Actualizar el contenido de **Prefix directory** con *$HOME/ISO_II/toolchain/${CT_TARGET}*. Este directorio se creará automáticamente con el nombre del target que se seleccionó previamente y es donde el toolchain será instalado.

   NOTA: Este path es importante dado que debe ser exportado para poder utilizar el toolchain una vez generado.
d. El valor Number of parallel jobs determina cuántos hilos (threads) se utilizan al invocar make en los distintos pasos de compilación. Mientras mayor sea este número, más recursos se dedicaran a la compilación, se recomienda no ejecutar otra aplicación a la par de ct-ng y utilizar el máximo de hilos disponibles para que la compilación sea lo más rápido posible, pero esto no es siempre posible. En el caso de estar tomando la clase en el momento de comenzar la compilación, se recomienda utilizar el 75% de los recursos para que no se produzca lag en la videollamada. El máximo número de hilos puede determinarse con el siguiente comando:
   ```
   $ cat /proc/cpuinfo | grep processor | wc -l
   ```
   Donde el archivo cpuinfo contiene toda la información del CPU, grep filtra las líneas que contengan la palabra processor y el comando wc con el flag -l (*word count --lines*) cuenta la cantidad de líneas que el comando grep acaba de filtrar. Se puede comprobar el número de procesadores prescindiendo del comando *wc*.

e. El valor **Maximum log level to see** debe estar seteado a **EXTRA**. En el caso de que exista un error el cual no puede ser determinado con este nivel de log, puede cambiarse a **DEBUG**. La cantidad de información de

salida para el valor **DEBUG** es demasiado grande para los casos de uso normales, y no se justifica utilizarlo.

2. Seleccionando la opción **Target options**, las opciones siguientes reflejan con más especificidad el hardware que se desea utilizar. En el caso de la materia, trabajando con BeagleBone Black:
    a. La opción **Architecture level** debe estar cargada con el valor **armv7-a**.
    b. La opción **Tune for CPU** debe estar cargada con el valor **cortex-a8**.
    c. La opción **Use specific FPU** debe estar cargada con el valor **neon**.
    d. La opción **Floating point** debe estar seteada en **hardware (FPU)**.
3. Seleccionando la opción **Toolchain options** se modifican algunos valores que tienen que ver con la invocación e información del toolchain:
    a. La opción **Toolchain ID string** contiene la identificación del toolchain. Puede cargarse una cadena de caracteres que incluya espacios. Para esta materia, actualizamos el valor a **MSE 7ta Cohorte**.
    b. La opción **Tuple's vendor string** contiene la identificación del *vendor*, y en este campo la cadena de caracteres no debería tener espacios. Se sugiere completar este campo con el valor **mse**.
    c. La opción **Tuple's alias** contiene un nombre corto con el que puede invocarse el compilador del toolchain una vez generado. Es muy útil dado que los nombres de los toolchains generados suelen ser largos. En definitiva lo que se crea es un link simbólico al compilador generado. Se recomienda completar este campo con el valor **arm-linux**. **NOTA:** Gracias al alias, la utilización de **arm-mse-linux-gnueabihf-gcc** (nombre generado automáticamente según las opciones dadas si se sigue esta guía de práctica) y de **arm-linux-gcc** es indistinta.
4. Seleccionando la opción **Debug facilities**, se completa la generación del toolchain con herramientas de depuración:
    a. Es necesario que la opción **gdb** esté habilitada.
    b. Dentro de **gdb**, las opciones **cross-gdb** y **Build a static gdbserver** son necesarias para poder hacer debug de las aplicaciones compiladas con el toolchain a generar.

c. Las herramientas **duma** y **ltrace** no son necesarias, se deja a criterio del lector incluirlas o no.

Explore las diferentes opciones disponibles navegando a través de los menús y consultando la ayuda para algunas opciones. No dude en consultar a su instructor por detalles disponibles en las opciones.

NOTA: *Es menester que el lector recuerde que las versiones de las **companion libraries** (GMP, MPFR, ISL, MPC) son dependientes de las versiones de **glibc** y de **binutils** utilizadas, teniendo interdependencia entre ellas y a su vez con la versión de **gcc** utilizada. No todas las combinaciones de versiones son compatibles entre sí, por lo que se recomienda utilizar los valores por defecto. Si las versiones por defecto son modificadas, cabe la posibilidad de que la compilación del toolchain no sea posible.*

Compilación del toolchain

Una vez finalizada la configuración de todo lo necesario, la compilación del toolchain es muy sencillo, solamente debe ejecutarse el comando build y esperar a que finalice.

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

Pasos finales

Prueba funcional

Para verificar que el toolchain fue generado de forma satisfactoria, puede ejecutarse el compilador con el flag *--version* y corroborar la salida del mismo. El paso previo es indicar al sistema operativo donde se encuentra este ejecutable, a menos que se corra el mismo desde el directorio que lo contiene. Según los pasos mostrados en esta guía, el path del cross-compiler generado será *$HOME/ISO_II/toolchain/arm-mse-linux-gnueabihf/bin*, el cual se puede navegar para ejecutar el compilador desde allí, o anexar a la variable **PATH** del sistema. Se recomienda hacer un **export** de esta variable para el bash utilizado cada vez que se desee trabajar compilado con este toolchain. Si bien el proceso es más tedioso, permite que el sistema no se “ensucie” con paths y herramientas que no son del mismo.

```
$ export PATH=$PATH:$HOME/ISO_II/toolchain/arm-mse-linux-gnueabihf/bin
```

NOTA: *La variable PATH exportada solo es visible dentro del bash actual (consola) cada vez que se inicie un bash nuevo, debe exportarse la variable nuevamente.*

Habiendo indicado al sistema donde está el compilador generado, puede ejecutarse el mismo con el flag *--version* y luego compilar un archivo .c trivial, como ser un *hello_world.c* para verificar que la compilación es exitosa.

Al ser un cross-compiler, el compilador es ejecutado en la estación de trabajo, pero genera binarios para la SBC, la cual tiene una arquitectura diferente. Por esta razón es imposible ejecutar en la estación de trabajo el binario generado, pero puede verificarse el tipo de archivo con el comando *file*.

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

$ file hello
```

Limpieza de archivos innecesarios

Para liberar aproximadamente 5 GB de espacio en disco, puede utilizarse el comando *clean* en el directorio de Crosstool-NG. Esta acción remueve el código fuente de diferentes componentes del juego de herramientas, como así también los archivos generados, que una vez instalado el toolchain, son innecesarios.

```
$ ./ct-ng clean
```