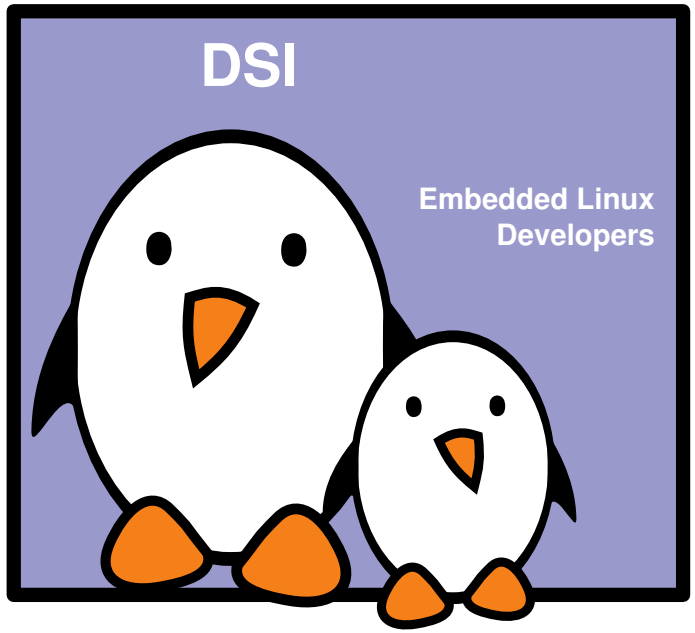
Juego de herramientas de compilación cruzada

Authors

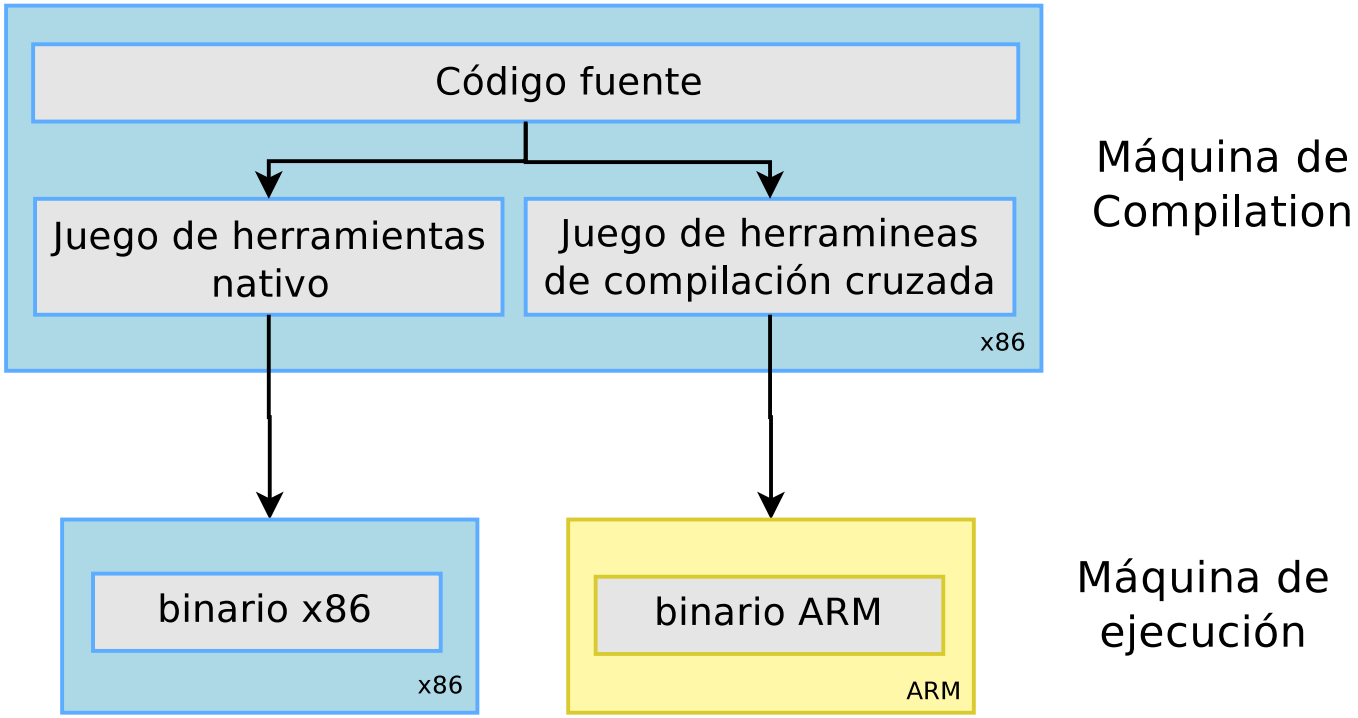


Definiciones y Componentes

Definiciones 1/2

- Las herramientas de desarrollo usualmente disponibles en una estación de trabajo GNU/Linux son un **juego de herramientas nativo**
- Este juego de herramientas corre en la estación de trabajo y genera código para la propia estación de trabajo (generalmente x86)
- Para el desarrollo de sistemas embebidos, es generalmente imposible o no tiene interes el utilizar un juego de herramientas nativo
  - El destino es muy restrictivo en terminos de almacenamiento y/o memoria
  - El destino es muy lento comparado con la estación de trabajo
  - Puede que no interese instalar todas las herraminetas de desarrollo en el destino.
- Por lo tanto, se utilizan generalmente **los juegos de herramientas de compilación cruzada**. Ellos corren en la estación de trabajo, pero generan código para el destino.

Definiciones 2/2



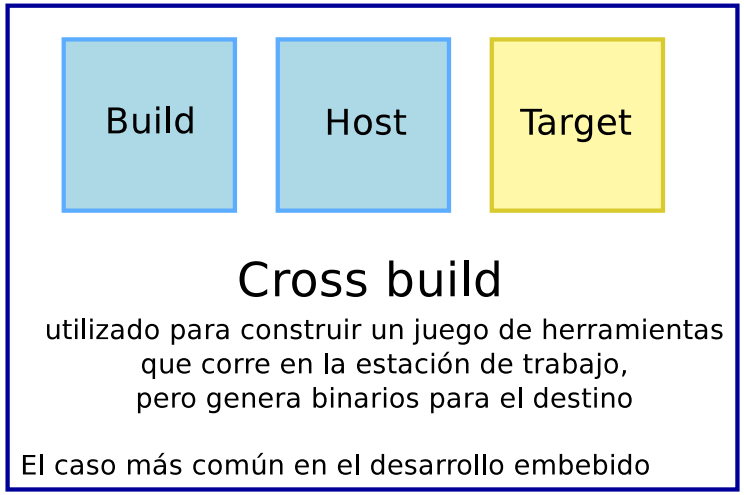
Máquinas en el proceso de construcción

- Se deben distinguir entre tres máquinas cuando se discute la creación del juego de herramientas
  - La máquina de construcción o **build**, donde se construye el juego de herraminetas.
  - La máquina **host**, donde se ejecuta el juego de herraminetas.
  - La máquina destino o **target**, donde los binarios creados por el juego de herraminetas son ejecutados.
- Cuatro formas de construcción son posibles para los juegos de herraminetas

Construcción del juego de herramientas

Native build

utilizado para construir el gcc normal
de una estación de trabajo



Cross build

utilizado para construir un juego de herramientas
que corre en la estación de trabajo,
pero genera binarios para el destino

El caso más común en el desarrollo embebido

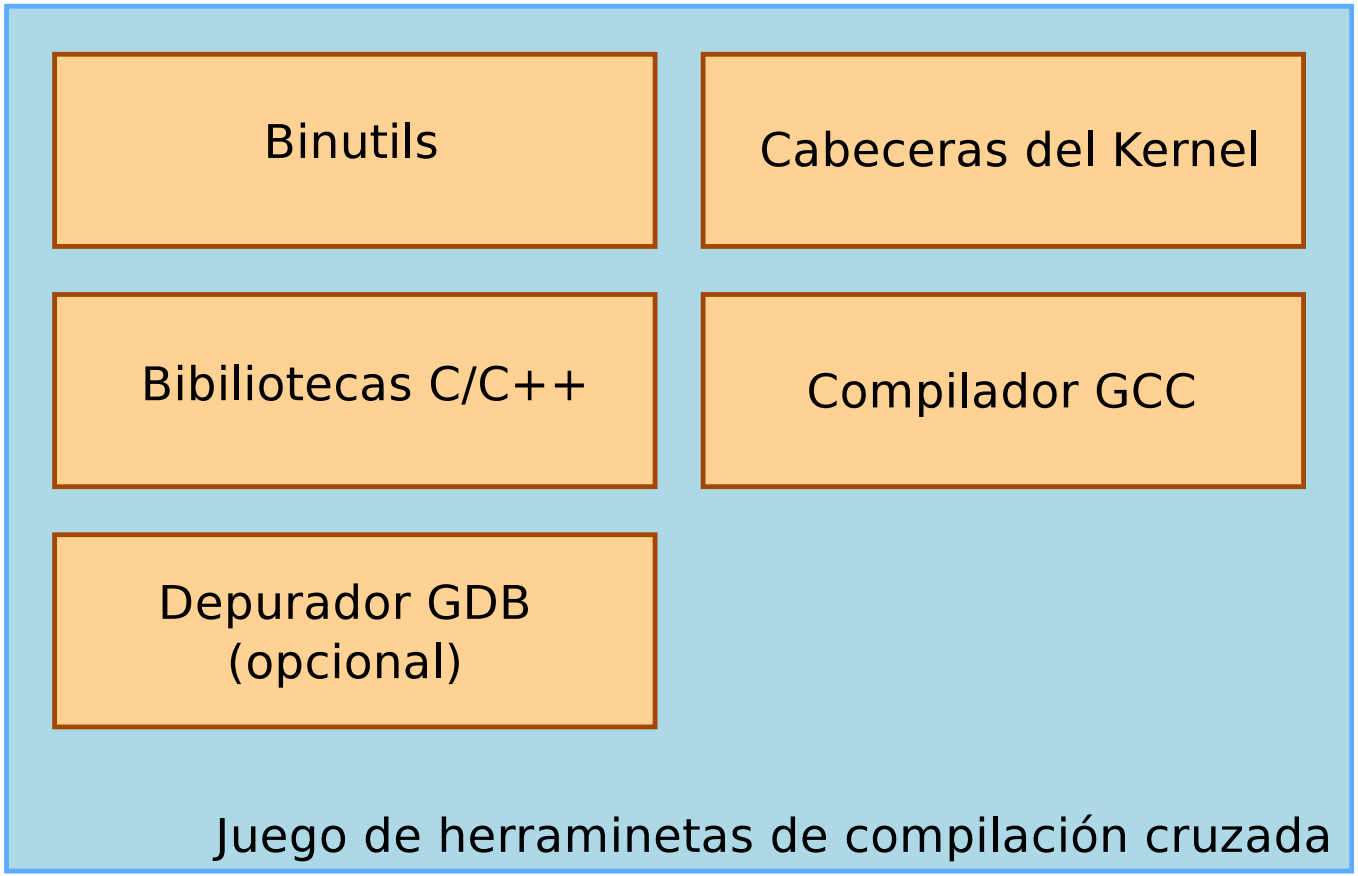
Cross-native build

utilizado para construir un juego de herraminetas que corra
en el destino y genere binarios para dicho destino

Canadian build

utilizado para construir en una arquitectura A un
juego de herramientas que corre en una arquitectura B
y genera binarios para la arquitectura C

Componentes

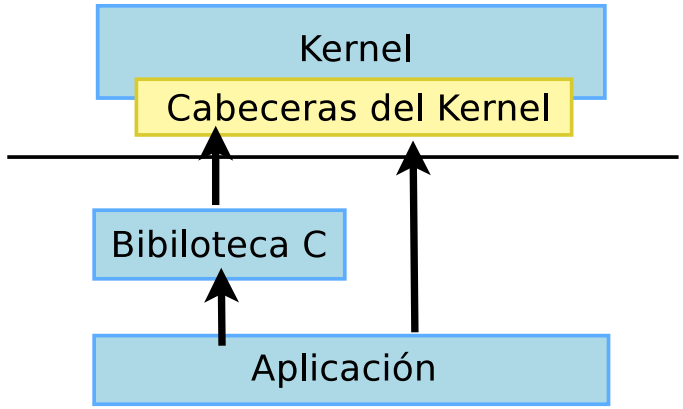


Binutils

- **Binutils** es un juego de herraminetas para generar y manipular binarios para una arquitectura CPU dada
  - `as`, el ensamblador, que genera código binario desde el código fuente ensamblador
  - `ld`, el enlazador
  - `ar`, `ranlib`, para generar los archivos `.a`, utilizados por las bibliotecas
  - `objdump`, `readelf`, `size`, `nm`, `strings`, para inspeccionar binarios. Herramientas de análisis muy útiles!
  - `strip`, para cortar partes de binarios inútiles y de esa forma reducir su tamaño
- `http://www.gnu.org/software/binutils/`
- Licencia GPL

Cabeceras del Kernel (1)

- La bibilioteca C y los programas compilados necesitan interactuar con el Kernel
  - Llamadas al sistema disponibles y sus correspondiente numeración
  - Definición de constantes
  - Estructuras de datos, etc.
- Por lo tanto, compilar la biblioteca C requiere de las cabeceras del Kernel, y varias aplicaciones también las requieren.
- Disponibles en los directorios `<linux/...>` y `<asm/...>` y en otros directorios correspondientes a los del directorio `include/` en los fuentes del Kernel



Cabeceras del Kernel (2)

- Números de llamadas al sistema, en <asm/unistd.h>

```
#define __NR_exit          1
#define __NR_fork          2
#define __NR_read          3
```

- Definiciones de constantes, en <asm-generic/fcntl.h>, incluidos desde <asm/fcntl.h>, incluidos desde <linux/fcntl.h>

```
#define O_RDWR 00000002
```

- Estructuras de datos, en <asm/stat.h>

```
struct stat {
    unsigned long st_dev;
    unsigned long st_ino;
    [...]
};
```

Cabeceras del Kernel (3)

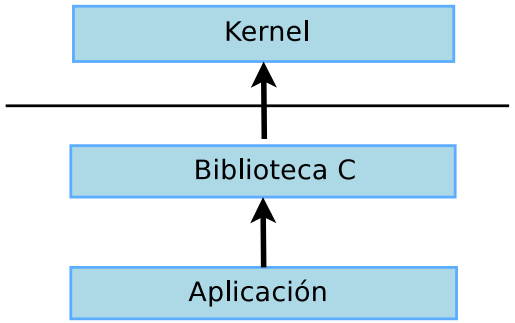
- El ABI del Kernel al espacio de usuario es **compatible hacia atras**
  - Binarios generados con un juego de herramientas utilizando cabeceras del Kernel mas viejas que las del Kernel en ejecución van a funcionar sin problemas, pero no se van a poder llamar a las nuevas llamadas al sistema, estructuras de datos, etc.
  - Binarios generados con un juego de herramientas utilizando cabeceras del Kernel más nuevas que las del Kernel en ejecucíon pueden funcionar si no utilizan las características recientes, de otra manera van a fallar
  - Utilizar las últimas cabeceras del Kernel no es necesario, salvo que se necesiten las nuevas características del Kernel
- Las cabeceras del Kernel se extraen del los fuentes del Kernel utilizando como destino del Makefile `headers_install`.

GCC

- GNU Compiler Collection, el famoso compilador de software libre
- Puede compilar C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, y generar código para un gran número de arquitecturas de CPU, incluyendo ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc.
- `http://gcc.gnu.org/`
- Disponible bajo la licencia GPL, bibliotecas bajo LGPL.

Bibioteca C

- La bibiloteca C es un componente escencial de un sistema GNU/Linux
  - Interactua entre la aplicación y el Kernel
  - Provee una API estandard de C bien conocida para facilitar el desarrollo de aplicaciones
- Hay disponibles varias bibliotecas de C: *glibc*, *uClibc*, *eglibc*, *dietlibc*, *newlib*, etc.
- La elección de la biblioteca C se debe realizar en el momento de la generación del juego de herramientas de compilación cruzada, dado que el compilador GCC se compila contra una biblioteca C específica.



Bibliotecas C

Opciones del Juego de herramientas

ABI (Application Binay Interface)

- Cuando se construye un juego de herramientas, el ABI utilizado para generar los binarios debe ser definido
- ABI define la convención de llamadas (como se pasan los argumentos a una función, como se retorna el valor, como se realizan las llamadas al sistema) y la organización de las estructuras (alineamiento, etc.)
- Todos los binarios en un sistema deben ser compilados con el mismo ABI, y el kernel debe entender dicho ABI.
- En ARM, existen dos ABIs: *OABI* and *EABI*
  - Actualmente se utiliza *EABI*
- En MIPS, existen varios ABIs: *o32, o64, n32, n64*
- `http://en.wikipedia.org/wiki/Application_Binary_Interface`

Soporte para punto flotante

- Algunos procesadores poseen una unidad de punto flotante, y otros no.
  - Por ejemplo, muchas CPUs ARMv4 y ARMv5 no tienen una unidad de punto flotante. Desde ARMv7, la unidad VFP (Vector Floating Point) es obligatoria.
- Para los procesadores que cuentan con una unidad de punto flotante, el juego de herramientas debe generar código *hard float*, y así poder utilizar las instrucciones de punto flotante directamente
- Para procesadores sin unidad de punto flotante, existen dos soluciones
  - Generar *hard float code* y dejar que el kernel emule las instrucciones de punto flotante (esto es muy lento).
  - Generar *soft float code*, para que en lugar de generar instrucciones de punto flotante, se generen llamadas a librerias en el espacio de usuario
- Esta decisión debe ser tomada en el momento de configuración del juego de herramientas

Banderas de optimización de CPU

- Un juego de herramientas para compilación cruzada es específico para una arquitectuda de CPU (ARM, x86, MIPS, PowerPC)
- Sin embargo, con la opción `-march=`, `-mcpu=`, `-mtune=`, se puede seleccionar de forma más precisa el tipo de CPU destino
  - Por ejemplo, `-march=armv7 -mcpu=cortex-a8`
- En el momento de compilar el juego de herramientas, se pueden elegir estos valores, los cuales serán utilizados:
  - Como valores por defecto para las herramientas de compilación cruzada, cuando ninguna otra opción `-march`, `-mcpu`, `-mtune` es utilizada
  - Para compilar la biblioteca C
- Incluso si la biblioteca C fue compilada para armv5t, no prohibe de compilar otros programas para armv7

Obteniendo el juego de herramientas

Construir el juego de herramientas de forma manual

Construir el juego de herramientas para la compilación cruzada uno mismo es una tarea dificil y compleja que puede llevar días e incluso semanas!

- Gran cantidad de detalles para aprender: muchos componentes, configuraciones complicadas
- Muchas decisiones que tomar (como ser la versión de la biblioteca C, ABI, mecanismos de punto flotante, versiones de los componentes)
- Son necesarios los fuentes de las cabeceras del Kernel y la biblioteca C
- Es necesario estar familiarizado con los problemas actuales y parches de `gcc` para su plataforma
- Es útil estar familiarizado con las herramientas de construcción y configuración
- Vea el directorio `docs/` de *Crosstool-NG* para detalles de como se construyen los juegos de herramientas.

Obtener un juego de herramientas pre-compilado

- Es una solución utilizada por muchas personas
  - Ventaja: es la solución más simple y la más conveniente
  - Desventaja: no es posible realizar un ajuste fino del juego de herramientas de acuerdo a sus necesidades
- Determinar que juego de herramientas necesita: CPU, endianism, biblioteca C, versión de componentes, ABI, soft float o hard float, etc.
- Verificar que los juegos de herramientas disponibles se ajustan a sus requerimientos.
- Opciones posibles
  - Juegos de herramientas de Sourcery CodeBench
  - Juegos de herramientas de Linaro
  - Más referencias en `http://elinux.org/Toolchains`

Sourcery CodeBench

- *CodeSourcery* fue una compañía con un gran conocimiento en juegos de herramientas libres: gcc, gdb, binutils y glibc. Fue adquirida por *Mentor Graphics*, la cual continúa brindando servicios y productos similares
- Venden juegos de herramientas con soporte, pero también poseen una versión "*Lite*", que es de libre y se puede utilizar en productos comerciales
- Tienen juegos de herramientas disponibles para
  - ARM
  - MIPS
  - PowerPC
  - SuperH
  - x86
- Asegúrese de utilizar versiones Linux. Las versiones EABI son para el desarrollo bare-metal (sin sistema operativo)

Juego de herramientas de Linaro

- Linaro contribuye a mejorar la línea principal de gcc para ARM, en particular contratando desarrolladores de CodeSourcery.
- Para aquellos que no pueden esperar por la próxima versión de gcc, Linaro lanza los fuentes modificados de versiones estables de gcc, con optimizaciones para ARM (en general para CPUs Cortex recientes).
- Como cualquier versión de gcc, estos fuentes pueden ser utilizados por las herramientas de construcción para construir sus propios juegos de herramientas binarios (Buildroot, OpenEmbedded, etc.) Esto permite soportar glibc, uClibc y eglibc.
- `https://wiki.linaro.org/WorkingGroups/ToolChain`
- Paquetes binarios están disponibles para usurios de Ubuntu, `https://launchpad.net/~linaro-maintainers/+archive/toolchain`

Instalar un juego de herramientas pre-compilado

- Siga el procedimiento de instalación propuesto por el fabricante
- Generalmente, es tan simple como extraer un archivo *tar* en la ruta donde quiera que quede instalado.
- Luego, agregue la ruta a los binarios del juego de herramientas en su `PATH`:
  `export PATH=/path/to/toolchain/bin/:$PATH`
- Finalmente, compile sus aplicaciones
  `PREFIX-gcc -o foobar foobar.c`
- `PREFIX` depende de la configuración del juego de herramientas, y permite distinguir entre herramientas de compilación cruzada y utilidades de compilación nativa

Utilidades para construir el juego de herramientas

Otra solución es utilizar utilidades que **automatizan el proceso de construcción del juego de herramientas**

- Las mismas ventajas que los juegos de herramientas pre-compilados: no necesita involucrarse en los detalles del proceso de construcción
- Pero también ofrece más flexibilidad en terminos de configuración del juego de herramientas, selección de la versión de componente, etc.
- Generalmente, también contienen varios parches que corrijen problemas conocidos en diferentes componentes en algunas arquitecturas
- Multiples herramientas con idénticos principios: shell scripts o Makefile que automáticamente recuperan, extraen, configuran, compilan e instalan los diferentes componentes

Utilidades para construir el juego de herramientas (2)

- **Crosstool-ng**
  - Refactorización de Crosstool, con una configuración similar al sistema menuconfig
  - Características: soporta uClibc, glibc, eglibc, hard float y soft float, varias arquitecturas
  - Mantenido activamente
  - `http://crosstool-ng.org/`

Utilidades para construir el juego de herramientas (3)

Varias utilidades para la construcción de sistemas de archivos raíz también permiten la construcción de juego de herramientas

- **Buildroot**
  - Basado en Makefile. Puede construir juegos de herramientas basado en (e)glibc, uClibc y musl, para una amplia gama de arquitecturas.
  - `http://www.buildroot.net`
- **PTXdist**
  - Basado en Makefile, uClibc o glibc, mantenido por *Pengutronix*
  - `http://pengutronix.de/software/ptxdist/`
- **OpenEmbedded / Yocto**
  - Muy completo en prestaciones, pero es un sistema de creación más complicado
  - `http://www.openembedded.org/`
  - `https://www.yoctoproject.org/`

Crosstool-NG: instalación y uso

- La instalación de Crosstool-NG se puede realizar a nivel del sistema o solamente de forma local en un directorio particular. Para la instalación local:

```
./configure --enable-local
make
make install
```

- Varias configuraciones de ejemplo para distintas arquitecturas están disponibles en samples, se pueden visualizar utilizando

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

- Para cargar una configuración de ejemplo

```
./ct-ng <sample-name>
```

- Para ajustar la configuración

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

- Para construir el juego de herramientas

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

Contenido del juego de herramientas

- Los binarios de compilación cruzada en `bin/`
  - Este directorio puede ser agregado a su `PATH` para utilizar de forma más simple el juego de herramientas
- Uno o más *sysroot*, cada uno conteniendo
  - La biblioteca C y bibliotecas relacionadas, compiladas para el destino
  - Las cabeceras de la biblioteca C y las cabeceras del Kernel
- Hay un *sysroot* por cada variante: los juegos de herramientas pueden ser *multilib* si poseen varias copias de la biblioteca C para diferentes configuraciones (por ejemplo: ARMv4T, ARMv5T, etc.)
  - Los juegos de herramientas de CodeSourcery para ARM son multilib, los sysroots están en `arm-none-linux-gnueabi/libc/`, `arm-none-linux-gnueabi/libc/armv4t/`, `arm-none-linux-gnueabi/libc/thumb2`
  - Los juegos de herramientas de Crosstool-NG pueden ser multilib también (todavía en etapa de experimentación), sino el sysroot está en

Laboratorio práctico - Utilizando Crosstool-NG

Es momento de construir el juego de herramientas

- Configure Crosstool-NG
- Ejecutelo para construir su propio juego de herramientas de compilación cruzada