

# Desarrollo de sistemas de linux embebidos

Mg. Ing. Gonzalo E. Sanchez  
MSE - 2022

**Implementación de Sistemas Operativos II**

# Desarrollo de sistemas de linux embebidos

- Introducción
- Hardware y arquitectura de sistemas embebidos
- Cross-Compile

# Introducción

# Introducción

- En 1983 Richard Stallman comienza el proyecto GNU y el desarrollo de gcc, gdb y glibc entre otras.
- En 1991 Linus Torvalds anuncia la primer versión de linux, un OS similar a UNIX utilizando las herramientas de Stallman.
- En 1995 Linux se torna más popular en servidores.
- En 2000 Linux se torna más popular en sistemas embebidos.
- En 2010 Linux se torna popular en dispositivos móviles.

# Introducción

- Cuando consideramos que un software es libre?
  - Libertad de correr el software para cualquier propósito
  - Libertad de estudiar el software y cambiarlo.
  - Libertad de redistribuir copias.
  - Libertad de distribuir copias de versiones modificadas.
- Todas estas libertades aplican a usos comerciales y no comerciales.
- Implican la disponibilidad de código fuente.

# Introducción

- La disponibilidad de código fuente hace que se pueda distribuir el producto y su código respectivo.
- Ideal para sistemas embebidos.
- **Definición:** Linux Embebido es el uso del Kernel de Linux y varios componentes open-source en un sistema embebido.

# Introducción

- Ventajas de la utilización de Linux y componentes open-source en sistemas embebidos:
  - Posibilidad de reutilizar componentes.
  - El ecosistema open-source provee muchos componentes para funcionalidades estándar (networking, gráficos, crypto, etc).
  - Tan pronto un HW o protocolo se difunde profusamente, hay alta probabilidad que exista un componente open-source soportandolos.
  - Permite el diseño y desarrollo de productos complicados basados en componentes existentes.
  - Permite focalizar esfuerzos en valor agregado del producto.

# Introducción

- El software gratuito permite ser duplicado en tantos dispositivos se quiera, libre de cargos.
- Costo de licencias para un dispositivo que solo utiliza open-source: \$0.
- Permite un mayor presupuesto para el desarrollo del hardware o para capacitación en torno a la problemática a abordar.
- Control total sobre el software que se incluye en el dispositivo y es parte del sistema.



# Introducción

- Gran ventaja: posibilidad de estudiar el software y decidir si es viable.
- Se pueden estudiar varias opciones antes de tomar una decisión.
- Permite explorar fácilmente nuevas posibilidades y soluciones.

# Introducción

- Punto fuerte: los componentes de software open-source son desarrollados por comunidades.
- Las comunidades ofrecen apoyo de alto nivel: contacto directo con los desarrolladores principales del componente.
- Las chances de obtener una respuesta no dependen de lo grande de la empresa donde se trabaje.
- Permite acelerar la resolución de problemas al desarrollar un sistema (Hard + Soft).

# Hardware y arquitectura de sistemas embebidos

# Hard & Arq

- Tanto el kernel como los componentes que dependen de la arquitectura soportan una amplia gama de la misma.
- Se soporta arquitecturas que posean MMU y también las que no.
- **ATENCIÓN:** Linux no está diseñado para microcontroladores pequeños.
- Fuera del set de herramientas, bootloader y el kernel, los demás componentes no dependen de la arquitectura.

# Hard & Arq

- Un sistema básico puede funcionar con:
  - 8MB de RAM.
  - 4MB de almacenamiento.
- Mínimo tamaño recomendado RAM 32MB.
- Se recomienda más espacio de almacenamiento (a criterio).
- Almacenamiento en Flash/NAND/NOR soportado.
- Almacenamiento en SD/MMC y eMMC soportado.

# Hard & Arq

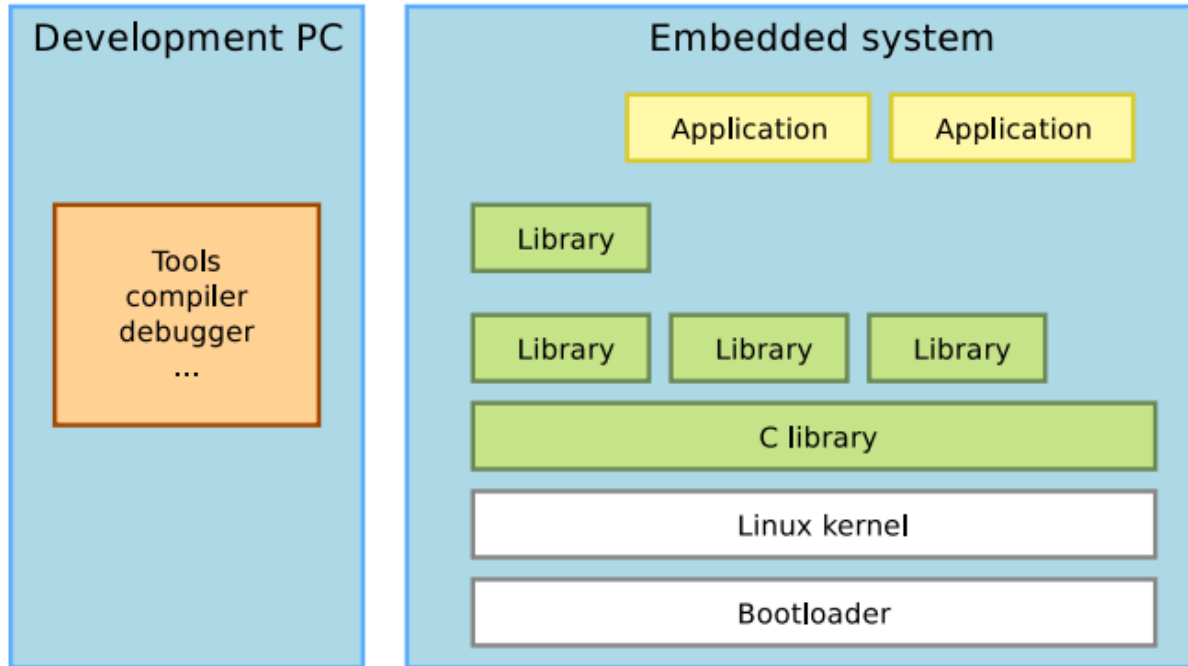
- Soporte del kernel para distintos buses:
  - I2C.
  - SPI.
  - USB.
  - CAN.
  - Entre otros.
- Soporte extensivo para redes (networking).

# Hard & Arq

- Criterios importante para la selección de Hardware a utilizar:
  - Soporte del kernel para el mismo.
  - Bootloader open-source que soporte el SoC destino.
- No todos los proveedores de SoC y/o plataformas contribuyen al kernel mainline.
- Una buena medida de la calidad de soporte es el delta existente entre el kernel mainline y el propietario.
- Existe una enorme diferencia de costos y tiempo de desarrollo entre HW soportado y HW sin soporte.

# Hard & Arq

- El siguiente diagrama muestra el set-up que tendremos para trabajar (ya conocido).





# Hard & Arq

- Es necesario un juego de compilación cruzada: corre en la plataforma host y compila para el target.
- Luego el BootLoader, inicialización básica y carga de Kernel.
- Kernel de linux. Manejo de hardware en general.
- C Library. Interfaz entre el kernel y el espacio usuario.
- Bibliotecas y aplicaciones en general.

# Hard & Arq

- Para distribuir Linux embebido en un dispositivo se necesitan ejecutar distintas tareas:
- **Board Support Package (BSP):** Contiene el bootloader y el kernel junto con los drivers correspondientes.
- **Integración del sistema:** Todos los componentes se unen para dar lugar a una workstation funcional.
- **Desarrollo de aplicaciones para Linux:** Aplicaciones comunes y corrientes que utilizan distintas bibliotecas.

# Hard & Arq

- Existen dos maneras de implementar una solución de linux embebido:
  - Utilizar una solución de un vendedor con soporte correspondiente.
  - Utilizar una solución de la comunidad, totalmente abiertas y soportadas.
- Todo el software de linux es provisto de una manera central y coherente: paquetes.
- Los paquetes tienen las bibliotecas o aplicaciones correspondientes e información acerca de dependencias.

# Hard & Arq

- Los paquetes están disponibles de distintos repositorios.
- Es una buena práctica utilizar solo repositorios oficiales para distribuir un producto en particular.
- En casos de fuerza mayor solamente se acepta la utilización de un repositorio no oficial.

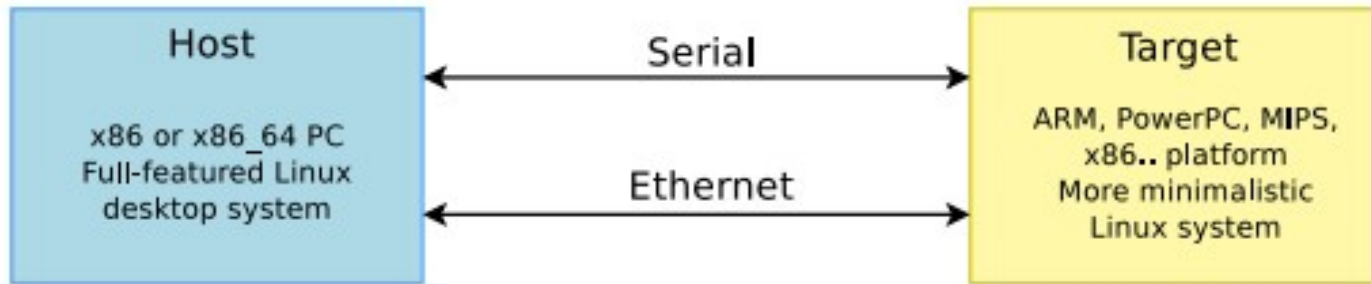
# Cross-Compile

# Cross-Compile

- Al trabajar con sistemas embebidos, siempre hay que hacer una división entre:
  - El sistema **HOST**: la estación de trabajo de desarrollo. Típicamente una PC de alto poder de proceso.
  - El sistema **TARGET**: el sistema embebido bajo desarrollo.
- Se conectan de distintas maneras, casi siempre mediante un puerto serie.
- Frecuentemente se utiliza una conexión ethernet y a veces una JTAG para debugging.

# Cross-Compile

- La herramienta esencial es la comunicación serie (picocom, Putty, gtkterm).
- Se utiliza la linea de comandos exclusivamente.
- Recordar el uso de [TAB] y de [CTRL]+[R].



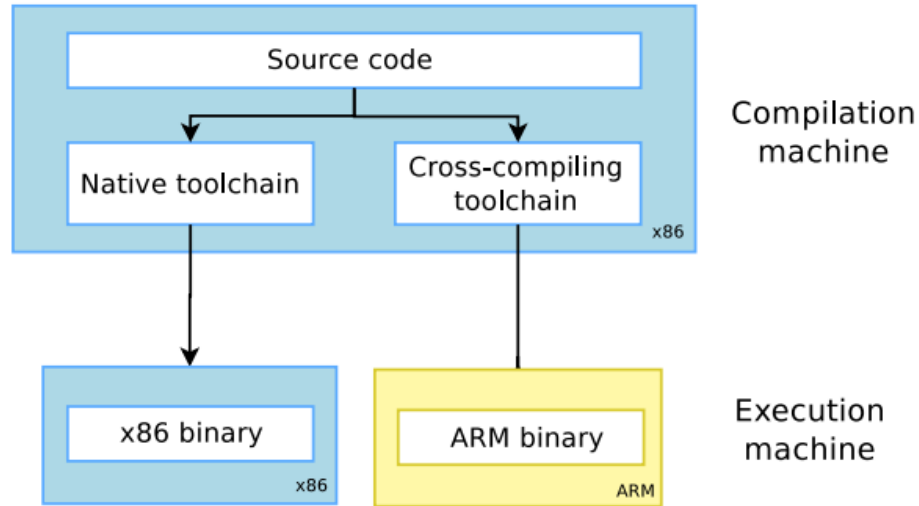
# Cross-Compile

- En cualquier terminal HOST se posee un compilador nativo (native toolchain).
- Corre en la estación de trabajo HOST y genera archivos ejecutables para la propia arquitectura (x86, amd64).
- Para el caso de los sistemas embebidos, no es común poseer herramientas de desarrollo:
  - El TARGET está restringido en memoria.
  - Es demasiado lento en comparación al HOST.
  - No es necesario instalar el juego de herramientas (no se utiliza).



# Cross-Compile

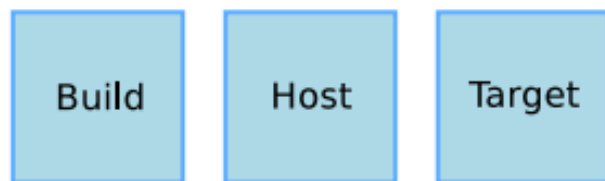
- Dado lo expuesto, lo general es utilizar un **cross-compile toolchain** (compilacion cruzada).
- Corre en el HOST y genera ejecutables para el TARGET.



# Cross-Compile

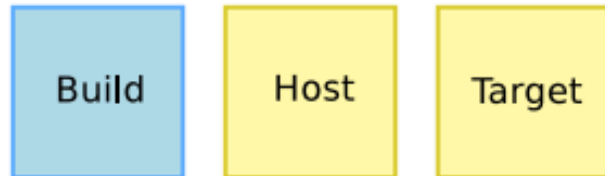
- Se distinguen 3 distintas máquinas al mencionar la creación de un toolchain:
  - BUILD: donde se construye el toolchain.
  - HOST: donde se ejecuta el toolchain.
  - TARGET: donde se corren los binarios producidos por el toolchain.
- Hasta ahora se han visto solamente HOST y TARGET.
- Son comunes 4 combinaciones de estas maquinas

# Cross-Compile



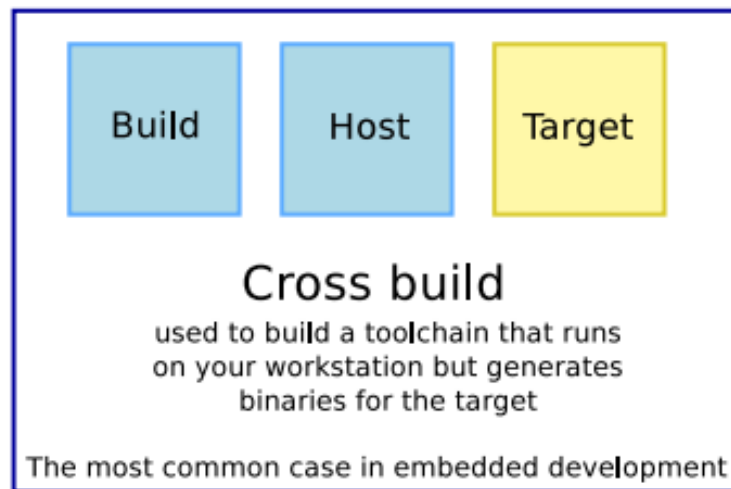
## Native build

used to build the normal gcc  
of a workstation



## Cross-native build

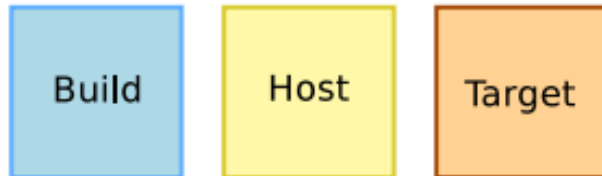
used to build a toolchain that runs on your  
target and generates binaries for the target



## Cross build

used to build a toolchain that runs  
on your workstation but generates  
binaries for the target

The most common case in embedded development

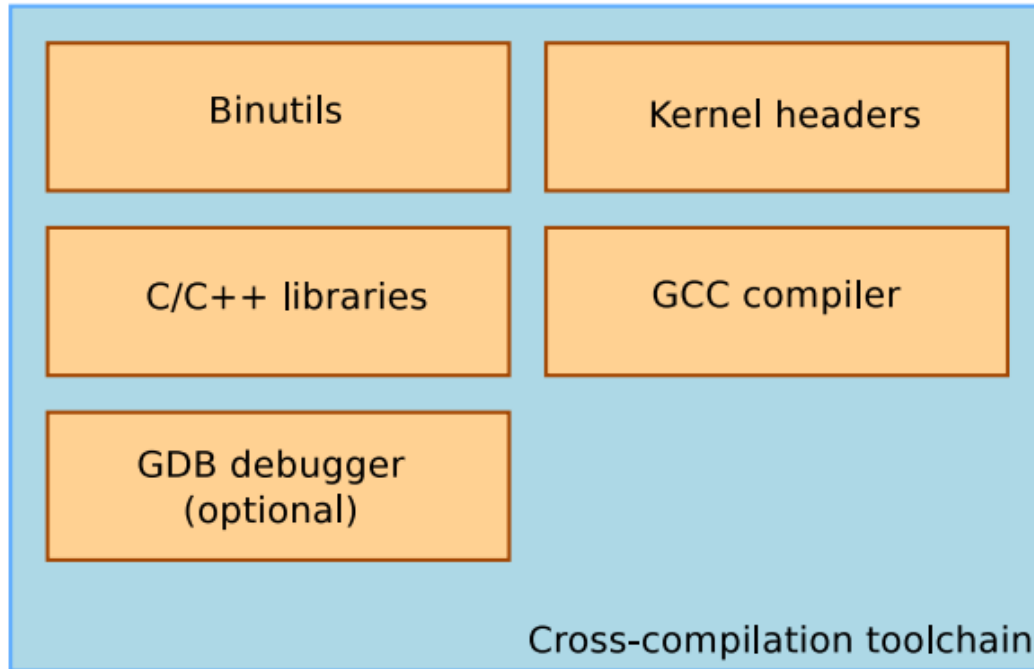


## Canadian build

used to build on architecture A a  
toolchain that runs on architecture B  
and generates binaries for architecture C

# Cross-Compile

- La gran pregunta es: por qué se denomina juego de herramientas (toolchain) y no solo compilador?

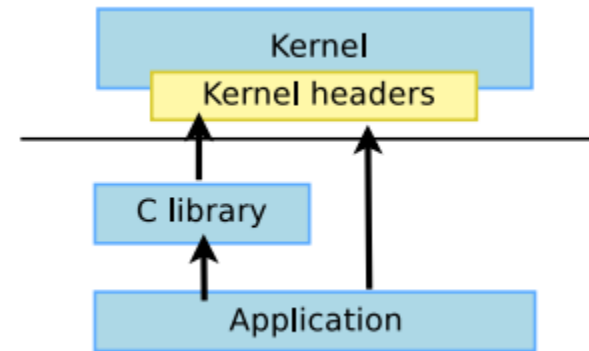


# Cross-Compile

- **Binutils** es un set de herramientas para generar y manipular binarios para una arquitectura CPU determinada.
  - **as** es el encargado de generar el binario a partir del assembler.
  - **ld** es el linker.
  - **ar, ranlib** encargados de generar los archivos **.a** para bibliotecas.
  - **objdump, readelf, size, nm, strings** son herramientas de análisis para binarios muy útiles.
  - **strip** se utiliza para quitar de los binarios información que no se utiliza y reducir su tamaño.

# Cross-Compile

- La biblioteca C y los programas compilados necesitan interactuar con el kernel.
- Deben conocer las system Calls disponibles, definiciones de constantes, estructuras de datos, entre otros.
- Para compilar la biblioteca C y algunos programas son necesarios los **headers del Kernel**.



# Cross-Compile

- La interfaz ABI entre el Kernel y el espacio usuario es compatible a versiones anteriores.
- Los binarios generados con headers anteriores a la versión de kernel que se está corriendo, funcionarían sin problema.
- Sin embargo no podrán utilizar las nuevas funcionalidades implementadas (claramente).
- Pero surge la duda: en IMD no se mencionó que la ABI **no** era estable? Módulos compilados con otros headers no funcionan!


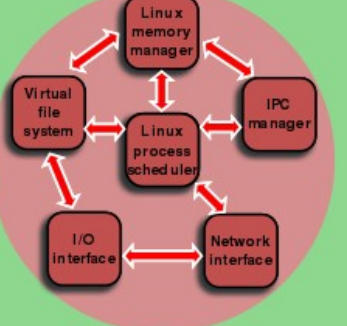
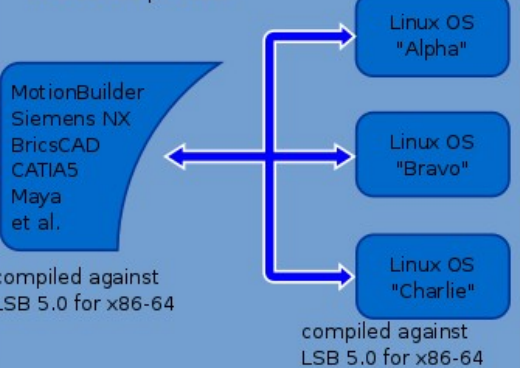
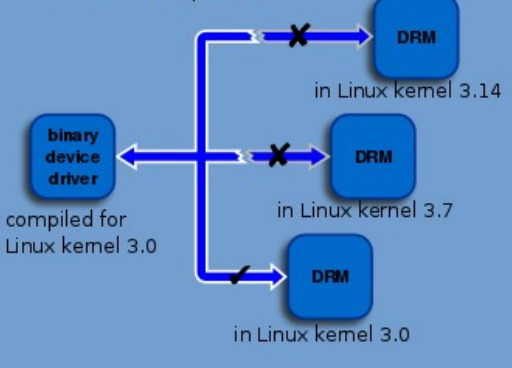
# Cross-Compile

- Esto es porque a ABI interna del kernel no es estable, pero la ABI kernel-espacio usuario si lo es.

*“We care about user-space interfaces to an insane degree. We go to extreme lengths to maintain even badly designed or unintentional interfaces. Breaking user programs simply isn’t acceptable.” – Linus Torvalds, 2005*



# Cross-Compile

	Linux kernel-to-userspace	Linux kernel-internal
API	<p>✓ API stability <b>is</b> guaranteed, source code is portable!</p> 	<p>✗ API stability <b>is not</b> guaranteed, source code portability is not given</p> 
ABI	<p>✓ compatible ABI <b>can be</b> guaranteed, binaries are portable</p>  <p>MotionBuilder Siemens NX BricsCAD CATIA5 Maya et al.</p> <p>compiled against LSB 5.0 for x86-64</p> <p>Linux OS "Alpha"</p> <p>Linux OS "Bravo"</p> <p>Linux OS "Charlie"</p> <p>compiled against LSB 5.0 for x86-64</p>	<p>✗ <b>no</b> stable ABI over Linux kernel releases, binaries are not portable</p>  <p>binary device driver</p> <p>compiled for Linux kernel 3.0</p> <p>DRM</p> <p>in Linux kernel 3.14</p> <p>DRM</p> <p>in Linux kernel 3.7</p> <p>DRM</p> <p>in Linux kernel 3.0</p>

# Cross-Compile

- GCC es el GNU Compiler Collection, el famoso compilador libre.
- Puede compilar C, C++, Ada, Fortran, Java, Objective-C, Objective-C++
- Genera código para ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86 64, IA64, Xtensa, etc
- Un compilador para todas las necesidades.

# Cross-Compile

- La biblioteca C es un componente esencial de un sistema Linux.
- Es la interfaz entre las aplicaciones y el kernel.
- Provee una muy conocida API en C estándar para el fácil desarrollo de aplicaciones.
- No existe una única biblioteca: glibc, uClibc, eglibc, dietlibc, newlib, etc.
- La elección se hace al momento de generar el cross compilador.

# Cross-Compile

- **glibc** posee licencia LGPL (GNU lesser general public license).
- Es la biblioteca C para el proyecto GNU.
- Está presente en todos los Sistemas GNU / Linux.
- Por supuesto, actualmente mantenida.
- Puede decirse que es grande para sistemas embebidos:
  - **libc**: 1.5 MB.
  - **libm**: 750KB.

# Cross-Compile

- **uClibc** posee licencia LGPL (GNU lesser general public license).
- Es la biblioteca peso ligero pensada para sistemas embebidos.
- Altamente configurable a traves de menuconfig.
- Solo funciona para Linux / uClinux.
- IMPORTANTE: No garantiza compatibilidad binaria. Puede ser necesario recompilar cuando la biblioteca se re-configura.
- El framework está en desarrollo, solo se han hecho pruebas

# Cross-Compile

- La mayoría de las aplicaciones compila con uClibc.
- El tamaño de la biblioteca para ARM es aproximadamente un 25% que el de glibc.
- Funcionalidades restringidas o directamente no presentes.
- Utilizada en muchos productos, incluyendo dispositivos de consumo masivo.

# Cross-Compile

- Madurez: soportada por todos los proveedores de linux embebido comerciales.
- **ATENCIÓN:** uClibc murió (last release 15 may 2012).
- Existe un fork que se anunció en OpenWRT en 2014: uClibc-ng
- uClibc-ng es actualmente mantenido (last release 06 jun 2020).

# Cross-Compile

- **eglibc** (Embedded glibc) posee licencia LGPL (GNU lesser general public license).
- Variante de glibc, no un fork. Intencionada a aceptar patches que no se aceptarían en glibc.
- Se desprendió de glibc por diferencias con los maintainers.
- Al cambiar el maintainer de glibc, y se incluyeron sus funcionalidades en glibc.
- Esto dio a que se descontinuara el proyecto (last release feb 2014).



# Cross-Compile

- Comparación de tamaños de binarios en ARM

<b>Programa Hola mundo</b>	<b>Estático</b>	<b>Dinámico</b>
<b>uClibc</b>	18 KB	2.5 KB
<b>uClibc con Thumb-2</b>	14 KB	2.4 KB
<b>eglibc con Thumb-2</b>	361 KB	2.7 KB

# Cross-Compile

- Comparación de tamaños de binarios en ARM

<b>BusyBox (stripped)</b>	<b>Estático</b>	<b>Dinámico</b>
<b>uClibc</b>	750 KB	603 KB
<b>uClibc con Thumb-2</b>	533 KB	439 KB
<b>eglibc con Thumb-2</b>	934 KB	444 KB

# Cross-Compile

- Existen otras bibliotecas C más pequeñas, ninguna tiene la meta de compilar proyectos grandes existentes.
- Es necesario escribir aplicaciones específicas para ellas.
- Ejemplos:
  - Dietlibc
  - Newlib
  - Klibc

# Cross-Compile

- Al construir un toolchain, el ABI debe estar definido.
- Implica convención de llamadas (como se pasan los argumentos a las funciones, valor de retorno, sysCalls).
- Todas las aplicaciones deben ser compiladas con el mismo ABI y el kernel entender ese ABI.
- Existen dos ABIs para ARM: **OABI** y **EABI**.
- OABI ya no se utiliza, asume una FPU presente en el core, no es el caso de ARM más antiguos.

# Cross-Compile

- OABI para ARM implicaba la virtualización de la FPU en el kernel, mediante manejo de excepciones.
- Implicaba cambios de contexto para cada instrucción FPU, haciendo las operaciones muy lentas.
- **EABI** soluciona esto emulando operaciones FPU en espacio usuario (no mas cambios de contexto).
- Al pasar los años, se incluyeron FPUs en los ARM cores.
- **EABIHF** se implementó, para casos VFP y NEON.

# Cross-Compile

- Un cross compilador es específico para una arquitectura (ejemplo ARM, x86, MIPS).
- Con algunos flags de configuración pueden seleccionarse el CPU destino.
- Ejemplo: **-march=armv7 -mcpu=cortex-a8**
- Compilar el juego de herramientas es una tarea ardua, y a veces tediosa.
- Muchos detalles para aprender, configuraciones difíciles, gran cantidad de componentes.

# Cross-Compile

- Gran cantidad de decisiones sobre versiones de biblioteca C, ABI a utilizar, mecanismo punto flotante, etc).
- Necesarios los headers del Kernel y biblioteca C (dependencias recursivas).
- Necesaria la familiarización con gcc, problemas y parches.
- Existen dos soluciones para facilitar el proceso:
  - Cross Compilador pre compilado (Sourcery CodeBench, GNU Arm toolchain antes distribuida por linaro).
  - Utilidades de automatizacion para compilación.

# Cross-Compile

- En esta materia utilizaremos una utilidad de automatización para compilar nuestro cross compiler.
- Ventajas: mismas que un paquete precompilado.
- Agrega flexibilidad para configuración fina.
- En general agregan parches para corregir problemas conocidos en componentes.
- Herramienta a utilizar: **Crosstool-ng** ([link](#))



# Cross-Compile

- Crosstools puede realizarse a nivel de sistema o bien en el directorio local.
- Es útil la confinación a un directorio local para no “ensuciar” el sistema.

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

# Cross-Compile

- Los binarios resultantes se guardan en la carpeta **/bin**.
- Recordar que la instalación es local, es necesario hacer el export del path a la variable del sistema \$PATH.
- Se define para cada juego de herramientas generado un *sysroot*.
- El sysroot puede pensarse como si fuera un directorio / pero dentro de un path determinado

# Desarrollo de módulos de kernel

## HANDS ON

1. Compilar un juego de herramientas con Crosstools-ng



Gracias.

