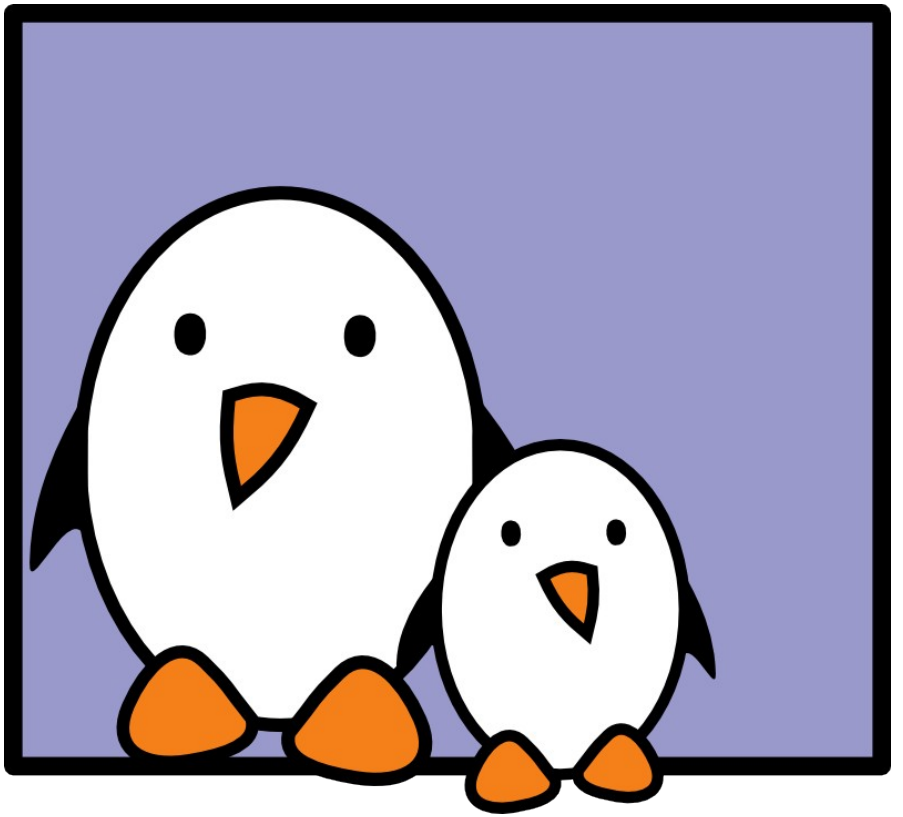
SASE

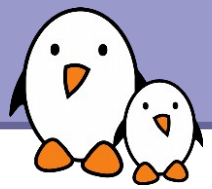
Simposio Argentino de Sistemas Embebidos

Workshop Linux Embebido



Lucas Chiesa
Joaquín de Andrés
Germán Bassi
Laboratorio
Sistemas embebidos
FIUBA

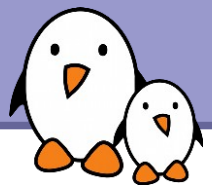
Basado en : http://free-electrons.com/docs/embedded-linux-intro



¿Sistema embebido?

Un sistema embebido o empotrado es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas frecuentemente en un sistema de computación en tiempo real. Los sistemas embebidos se utilizan para usos muy diferentes a los usos generales a los que se suelen someter a las computadoras personales.

Wikipedia, http://es.wikipedia.org/wiki/Sistema_embebido



Muchos sistemas diferentes

Es una definición muy genérica:

- Cubre muchos tipos diferentes de sistemas
- Línea borrosa con sistemas tradicionales

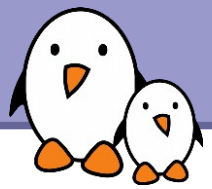
Productos “Consumer Electronics (CE)”:

- Routers hogareños, reproductores de DVD, Televisores, cámaras digitales, GPS, celulares ...

Productos industriales:

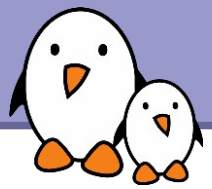
- Controladores de máquinas, alarmas, equipos de vigilancia, autos, satélites...

Muchos productos diferentes



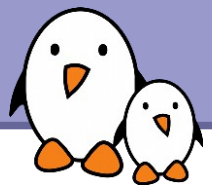
Linux embebido

- El Software Libre y Abierto ofrece una rango muy amplio de herramientas para desarrollar sistemas embebidos.
- Ventajas
  - Reutilizar componentes existentes para el sistema base. Permite concentrarse en el valor agregado del producto.
  - Componentes de alta calidad y muy probados. (Kernel Linux , librerías de C ...)
  - Control completo sobre la elección de componentes. Modificaciones posibles ilimitadas.
  - Soporte por la comunidades: tutoriales, listas de correo...
  - Bajo costo, sin licencias por unidad.
  - Acceso más simple al software y a las herramientas.

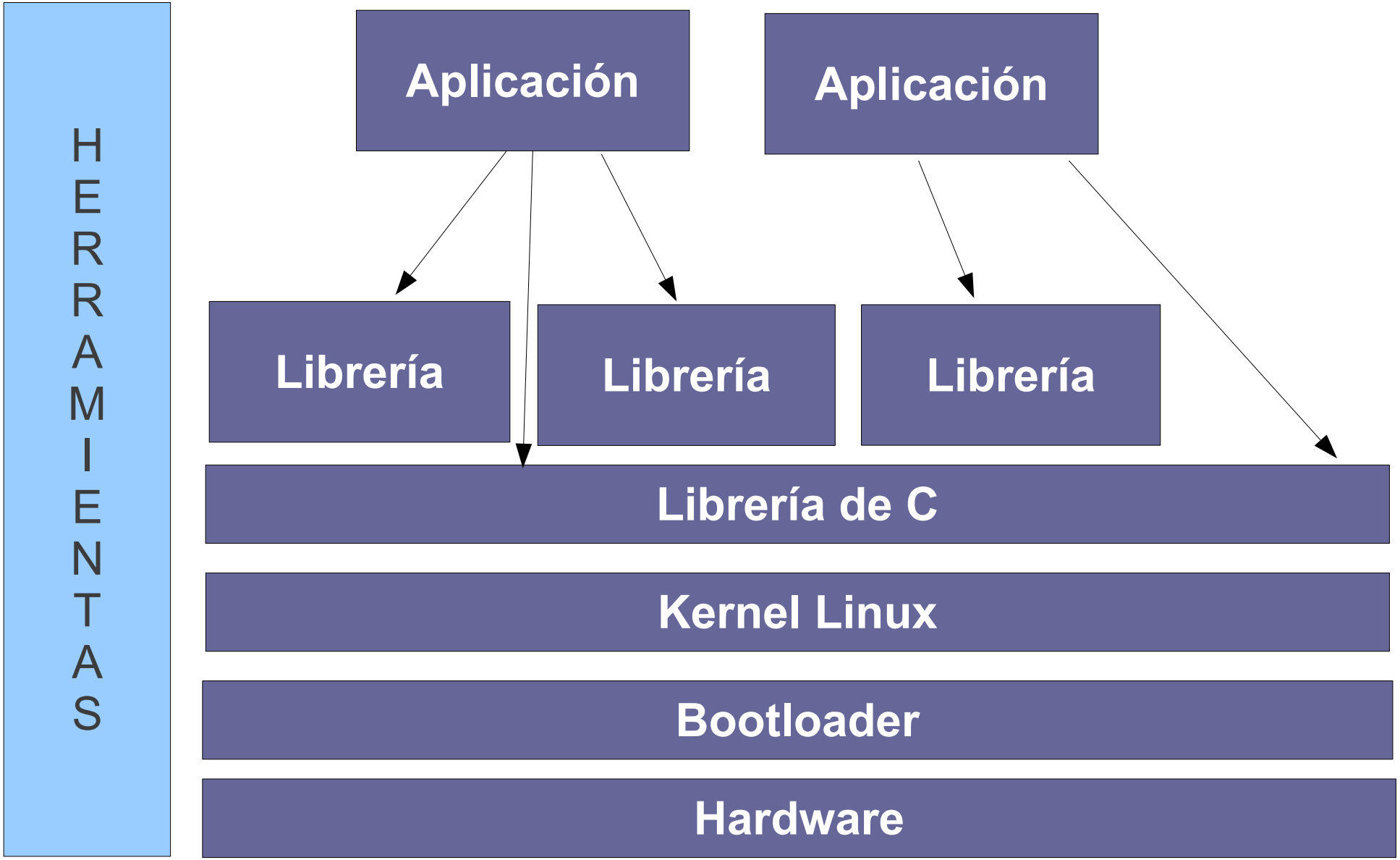


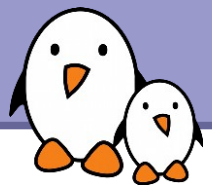
Ejemplos de dispositivos

- GPS: TomTom y Garmin
- Routers hogareños: Linksys, Netgear
- PDA: Zaurus, Nokia N8x0
- TVs, DVDs: Sony, Philips, ...
- Celulares: Motorola, Android, OpenMoko
- Y muchos productos que uno no se imagina...



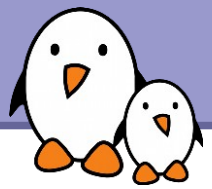
Arquitectura global





Hardware embebido

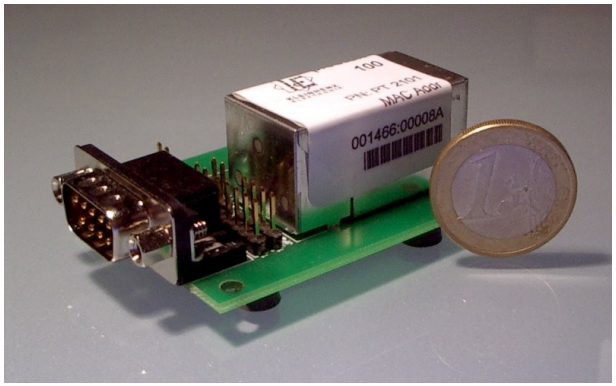
- El hardware para sistemas embebidos es usualmente diferente de los sistemas comunes.
  - Normalmente usan otras arquitecturas:
    ARM, MIPS o PowerPC. x86 también es usado.
  - Almacenamiento flash (NOR o NAND), usualmente con capacidad limitada. (de pocos MB a cientos de MB)
  - Capacidad de RAM limitada (desde unos pocos MB)
  - Muchos buses de conexión no comunes en desktops: I2C, SPI, 1-wire, CAN, etc.
- Placas de desarrollo por cientos de dolares
  - Usualmente usadas como base de nuestro diseño.



Ejemplos

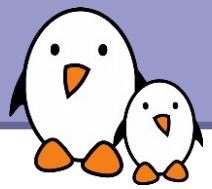
Picotux 100

- ARM7 55 MHz, Netsilicon NS7520
- 2 MB de flash
- 8 MB de RAM
- Ethernet
- 5 GPIOs
- Serial



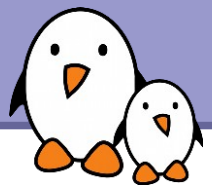
BeagleBoard C4

- Cortex A8 750 MHz, TI OMAP 3550
- 256 MB de RAM
- 256 MB de NAND flash
- DVI, audio entrada y salida, USB, etc



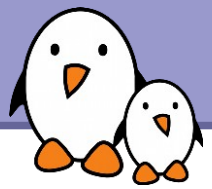
Requerimientos mínimos

- Una CPU soportada por el GCC y el Kernel Linux
  - 32 bit CPU
  - CPUs sin MMU son soportadas por uClinux, que no vamos a ver ahora.
- Unos MB de RAM, desde 4 MB.
  8 MB para hacer algo útil.
- Unos pocos MB de almacenamiento, desde 2 MB.
  4 MB para hacer algo útil.
- Linux no está diseñado para procesadores con KBs de RAM o ROM:
  - Se usan sin SO.
  - Sistemas reducidos, como FreeRTOS



Componentes de software

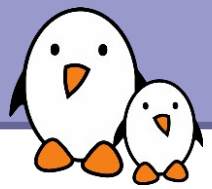
- Cross-compilation toolchain
  - Compilador que se ejecuta en una computadora de desarrollo y genera binarios para el destino.
- Bootloader
  - Iniciado por el hardware, es el responsable de la inicialización básica y ejecutar el Kernel.
- Kernel Linux
  - Contiene el administrador de procesos y memoria, stack de red, controladores de dispositivos, provee servicios a las aplicaciones.
- Librería de C
  - Interfaz entre el Kernel y las aplicaciones de usuario.
- Librerías y aplicaciones
  - Propias o producidas por otros.



Armado de un sistema Linux

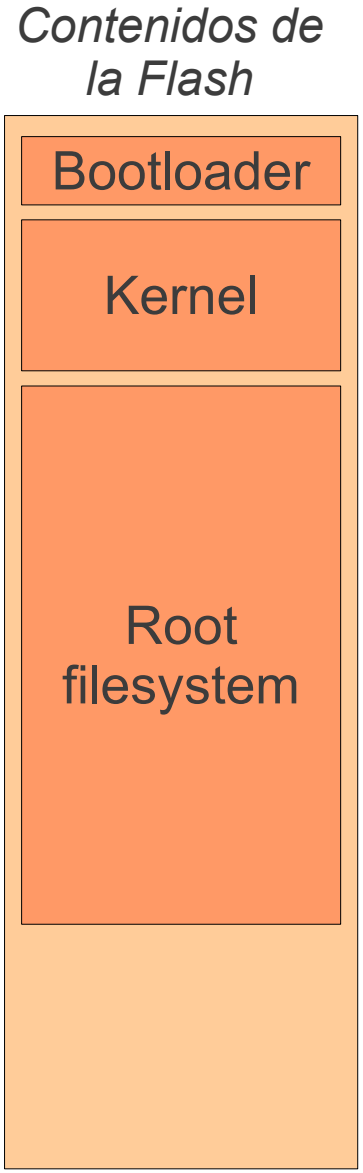
Se necesitan realizar diferentes tareas para producir un sistema Linux embebido:

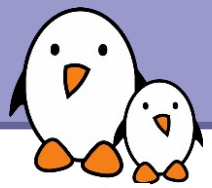
- **Desarrollo del Board Support Package**
  - Un BSP contiene el bootloader y Kernel con los drivers necesarios para mi hardware destino.
  - No es donde nos vamos a centrar.
- **Integración del sistema**
  - Integrar todos los componentes, BSP, librerías y aplicaciones utilizadas y las aplicaciones propias para obtener un sistema funcional.
  - Propósito de este curso.
- **Desarrollo de aplicaciones**
  - Aplicaciones normales de Linux, pero usando librerías especiales.



Root filesystem

- En un sistema Linux, los sistemas de archivos se “montan” dentro de una jerarquía global de directorios y archivos.
- Un sistema particular, el **root**, se monta como el directorio /.
- En sistemas embebidos, este sistema de archivos contiene todos los archivos del sistema.
- Por lo tanto, crearlo es una de las tareas principales de la integración de componentes.
- El Kernel normalmente se mantiene separado.

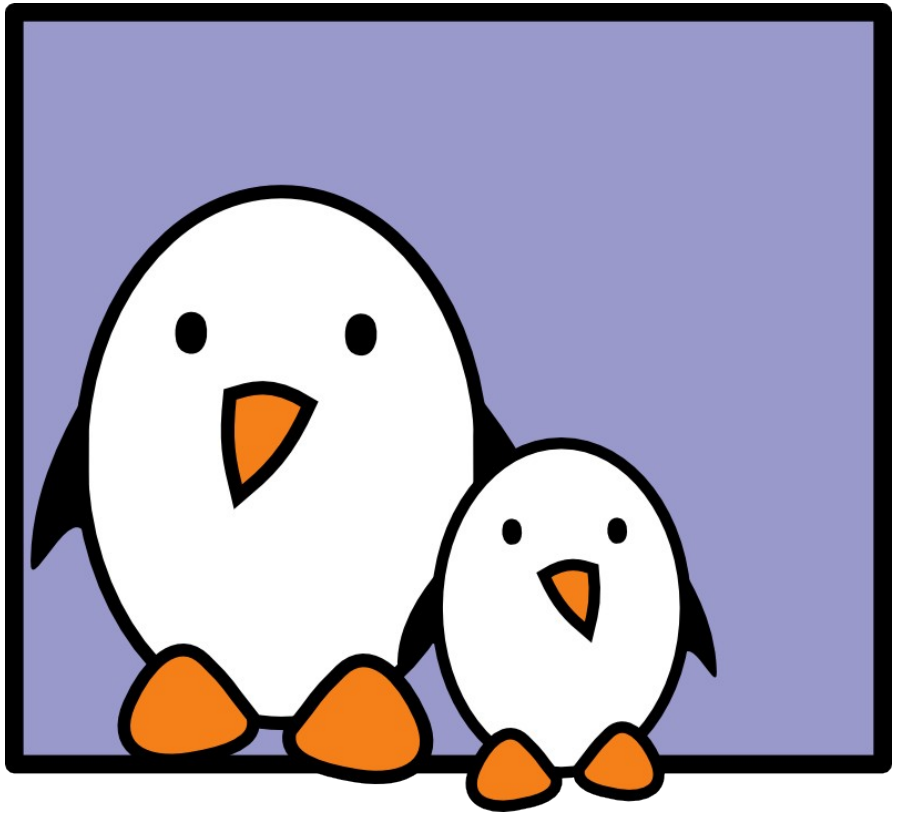


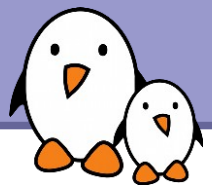


Entorno de desarrollo

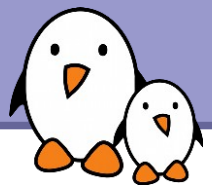
- Dos maneras de desarrollar un sistema Linux:
  - Usar soluciones provistas y soportadas por empresas como MontaVista, Wind River o TimeSys. Cada una tiene sus herramientas propias.
  - Usar herramientas provistas por la comunidad.
- No vamos a promover ninguna empresa en particular, por lo que vamos a usar herramientas de la comunidad.
  - Lo importante es entender los conceptos, migrar de herramienta siempre es más fácil.
- Desarrollar un sistema Linux **requiere** usar Linux.
  - Las herramientas comunitarias no están disponibles para otras plataformas.
  - Entender Linux en el desktop nos va a facilitar el desarrollo de Linux embebido.

Uso de un sistema Linux





Sistema de archivos y dispositivos



Creando un sistema de archivos

Ejemplos

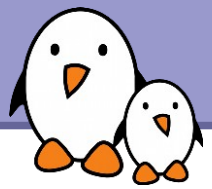
- `mkfs.ext2 /dev/sda1`
  Formatea la partición `/dev/sda1` en formato ext2.
- `mkfs.ext2 -F disk.img`
  Formatea una imagen de disco en ext2

`-F`: force. Ejecuta aunque no sea un dispositivo real.

- `mkfs.vfat -v -F 32 /dev/sda1` (`-v`: verbose)
  Formatea la misma partición anterior en FAT32.
- `mkfs.vfat -v -F 32 disk.img`
  Formatea la imagen en FAT32.

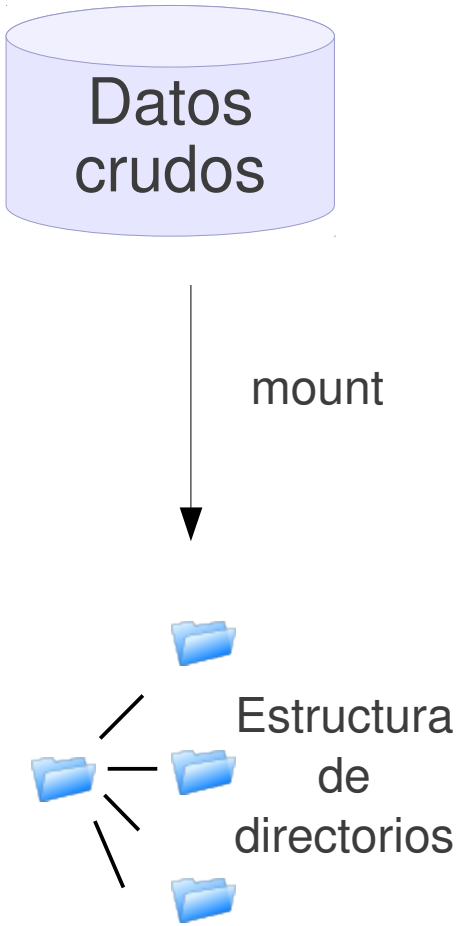
Imágenes de discos en blanco se pueden crear así: (archivo de 64 MB):

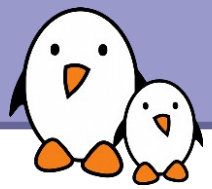
```
dd if=/dev/zero of=disk.img bs=1M count=64
```



Montando un dispositivo

- Para hacer cualquier sistema de archivos visible en el sistema hay que *montarlos*.
- La primera vez, se crea el punto de montaje: `mkdir /mnt/usbdisk` (ejemplo)
- Ahora lo montamos:
  `mount -t vfat /dev/sda1 /mnt/usbdisk`
  `/dev/sda1`: dispositivo físico
  `-t`: especifica el formato del sistema de archivo (`ext2`, `ext3`, `vfat`, `reiserfs`, `iso9660`...)



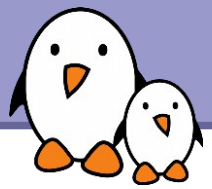


Montando un dispositivo

También se puede montar una imagen de un sistema de archivos guardada en un archivo regular (*loop devices*)

- Muy útil para desarrollar y probar un sistema de archivos que estamos armando para otra computadora.
- Útil para acceder a los contenidos de una imagen de CD (iso) sin tener que grabar el disco.
- Ejemplo:

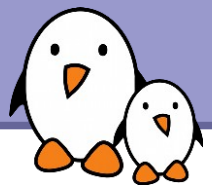
```
mount -o loop -t vfat usbkey.img /mnt/usbdisk
```



Listando dispositivos montados

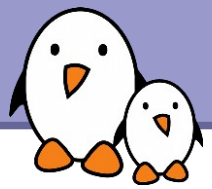
Usando el comando `mount` sin argumentos se obtiene la lista:

```
/dev/hda6 on / type ext3 (rw,noatime)
none on /proc type proc (rw,noatime)
none on /sys type sysfs (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda4 on /data type ext3 (rw,noatime)
none on /dev/shm type tmpfs (rw)
/dev/hda1 on /win type vfat (rw,uid=501,gid=501)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
```



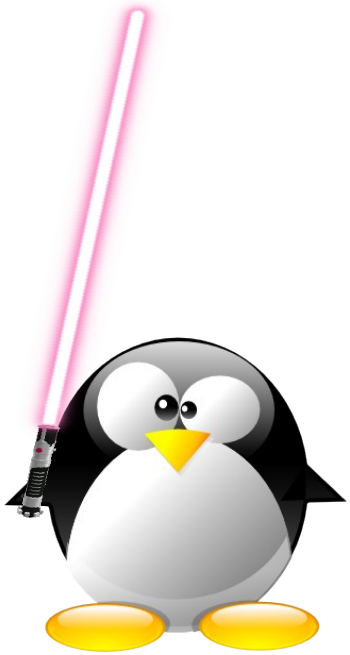
Desmontando dispositivos

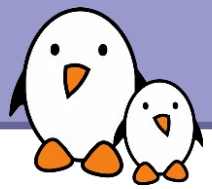
- `umount /mnt/usbdisk`
  Termina todas las transacciones pendientes en el dispositivo y lo desmonta.
- Para poder desmontar un dispositivo hay que cerrar todos sus archivos abiertos:
  - Cerrar todos los programas que usen algún archivo del directorio montado.
  - Estar seguro que ningún shell esté abierto en esa partición.
  - Se puede usar el comando `lsof <mount point>` (list open files) para ver todos los archivos abiertos en ese directorio.



El lado oscuro de root

- `root:` Se necesitan sus privilegios sólo para operaciones específicas, con impacto en la seguridad. Por ejemplo: montar, crear dispositivos, cargar drivers, configurando la red, cambiar permisos, instalar paquetes...
- Por más que uno tenga la clave de `root`, la cuenta normal tiene que ser suficiente para el 99.9% de las actividades (a menos que seas el sysadmin)
- En este curso es aceptable que usen `root`. En la vida real, pueden no tener esta cuenta disponible.

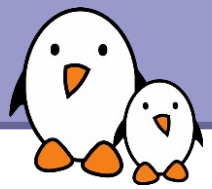




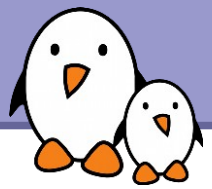
Usando la cuenta root

En caso que uno realmente quiera ser `root`...

- Si uno tiene el password de `root`:
  `su -` (**s**witch **u**ser)
- En distribuciones modernas, el comando `sudo` te da acceso a algunos privilegios de `root` usando la clave de tu usuario.
  Ejemplo: `sudo mount /dev/hda4 /home`

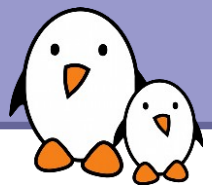


Manejo de paquetes



Paquetes de software

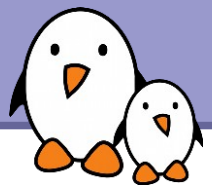
- La forma de distribuir software en GNU/Linux difiere de la que se usa en Windows.
- Las distribuciones de Linux proveen una forma central y coherente de instalar, actualizar y borrar aplicaciones y librerías: **Paquetes**.
- Los paquetes contienen la aplicación, las librerías e información extra, como la versión y las dependencias.
  - .deb en Debian y Ubuntu, .rpm en Mandriva, Fedora, OpenSUSE
- Los paquetes se guardan en **repositorios**, usualmente servers HTTP o FTP.
- Siempre hay que usar paquetes oficiales de la distribución, a menos que sea estrictamente necesario.



Manejando paquetes de software

Instrucciones para sistemas basados en Debian GNU/Linux: (Debian, Ubuntu...)

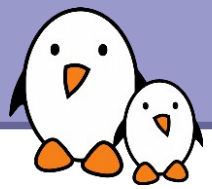
- Los repositorios de paquetes se especifican en:
  `/etc/apt/sources.list`
- Para actualizar la lista de paquetes:
  `sudo apt-get update`
- Para encontrar un paquete para instalar, se pueden usar los buscadores web: http://packages.debian.org o http://packages.ubuntu.com. También se puede usar:
  `apt-cache search <keyword>`



Manejando paquetes de software

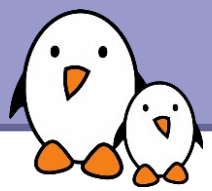
- Para instalar un paquete:
  `sudo apt-get install <package>`
- Para desinstalar un paquete:
  `sudo apt-get remove <package>`
- Para aplicar todas las actualizaciones:
  `sudo apt-get dist-upgrade`
- Para obtener información de un paquete:
  `sudo apt-cache show <package>`
- Existen interfaces gráficas:
  - Synaptic para GNOME
  - Adept para KDE

Para más información sobre administración de paquetes:
http://www.debian.org/doc/manuals/apt-howto/



Apagando el sistema

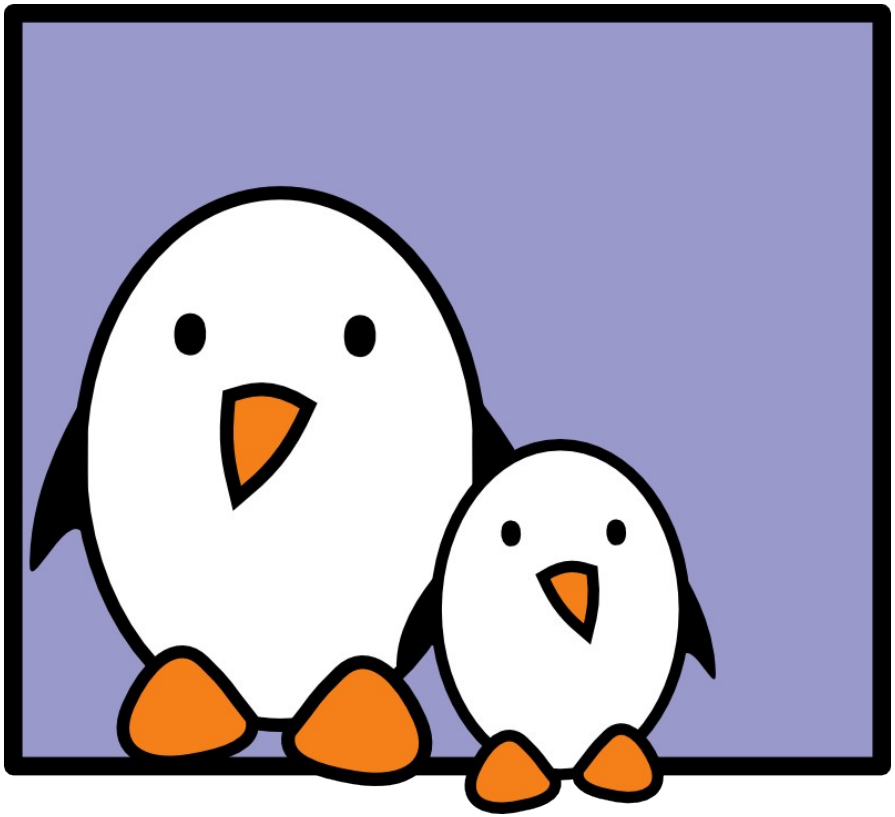
- `halt`
  Apaga inmediatamente el sistema.
- `reboot`
  Reinicia inmediatamente el sistema.
- `[Ctrl][Alt][Del]`
  También funciona el GNU/Linux para reiniciar.
  Sistemas embebidos: hay que usar un `init` y especificar una combinación de teclas en `/etc/inittab`.

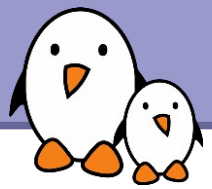


Práctica – Usando Linux

- Practicar comandos básicos de Linux.
- Familiarizarse con la sintaxis del shell.
- Perderle el miedo a la terminal.

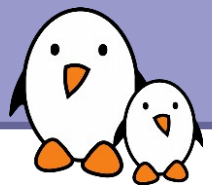
Introducción al Kernel Linux



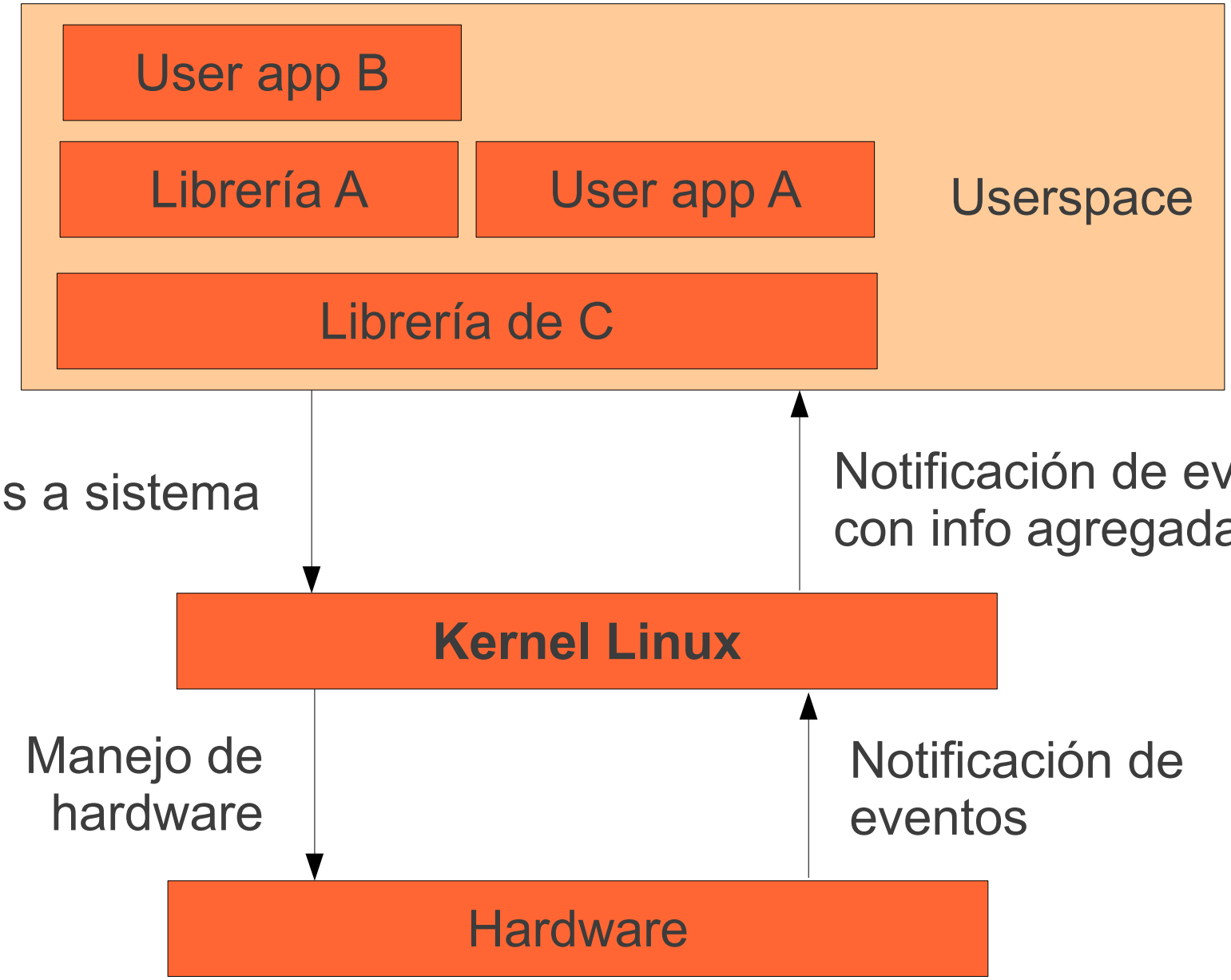


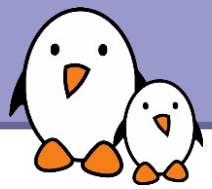
Presentando el kernel

Características de Linux



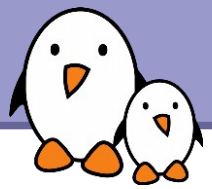
Kernel Linux en el sistema





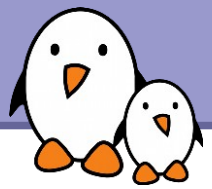
Historia

- El Kernel Linux es un componente del sistema, requiere librerías y aplicaciones para proveer funcionalidad a los usuarios finales.
- Creado por un estudiante finlandes, Linus Torvalds, como un hobby en 1991.
  - Rápidamente fue adoptado como el Kernel para formar un sistema operativo libre.
- Linus Torvalds fue capaz de crear una comunidad de usuarios y programadores grande y dinámica alrededor de Linux.
- Actualmente cientos de personas contribuyen a cada release del Kernel, tanto individuos particulares como pertenecientes a grandes empresas.



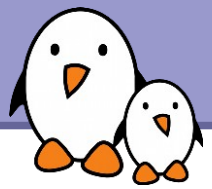
Licencia de Linux

- Todas las fuentes del Kernel son software libre, liberado bajo la licencia GNU General Public License version 2 (GPL v2).
- Para el Kernel, esto implica que:
  - Cuando uno recibe o compra un dispositivo con Linux dentro, debe recibir las fuentes de Linux, con todos los derechos de estudiar modificar y redistribuir.
  - Cuando uno produce equipos basados en Linux, uno debe dar las fuentes al destinatario, sin restricciones.
- Más adelante vamos a ver en detalle las licencias libres.



Características principales

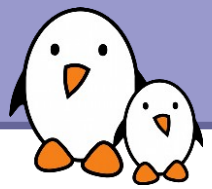
- Portabilidad y soporte de hardware. Corre en la mayoría de las arquitecturas.
- Escalabilidad.
  Funciona en super computadoras como en pequeños dispositivos.
  (4 MB de RAM es suficiente).
- Cumple con estándares de interoperatividad
- Muy completo y robusto soporte de red.
- Seguridad
  No puede ocultar las fallas. El código es revisado por muchos expertos.
- Estable y confiable.
- Modular. Puede incluir sólo lo que nuestro sistema necesita.
- Fácil de programar. Se puede aprender de código existente. Muchos recursos en Internet



Arquitecturas soportadas

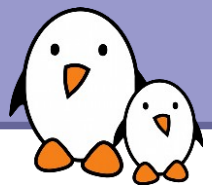
Para el Kernel 2.6.31

- Revisar el directorio `arch/` en las fuentes.
- Requerimiento mínimo: 32 bits, con o sin MMU, y soporte de gcc.
- Arquitecturas de 32 bits
  `arm`, `avr32`, `blackfin`, `cris`, `frv`, `h8300`, `m32r`, `m68k`, `m68knommu`, `microblaze`, `mips`, `mn10300`, `parisc`, `s390`, `sparc`, `um`, `xtensa`
- Arquitecturas de 64 bits:
  `alpha`, `ia64`, `sparc64`
- Arquitecturas de 32 o 64 bits:
  `powerpc`, `x86`, `sh`
- Más detalles en: `arch/<arch>/Kconfig`, `arch/<arch>/README`, o `Documentation/<arch>/`



Llamadas a sistema

- La principal interfaz entre el Kernel y userspace es un conjunto de llamadas a sistema.
- Existen ~300 llamadas a sistemas que proveen los servicios principales del Kernel
  - Operaciones en archivos y dispositivos, operaciones de red, comunicación entre procesos, administración de procesos, manejo de memoria, timers, threads, sincronización, etc.
- Interfaz estable en el tiempo: sólo se pueden agregar nuevas llamadas.
- La interfaz a las llamadas al sistema es abstraída por la librería de C. Los programas de userspace usualmente no hacen llamadas al sistema, usan las funciones correspondiente de la librería.



Sistemas de archivos virtuales

- Linux ofrece información del sistema y el Kernel a userspace mediante sistemas de archivos virtuales (virtuales: no existen en un ningún almacenamiento real). No se necesita saber programar para acceder a esta información!
- Montar `/proc`:

```
sudo mount -t proc none /proc
```

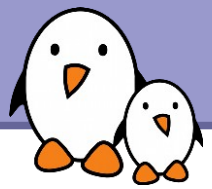
- Montar `/sys`:

```
sudo mount -t sysfs none /sys
```

Tipo de fs

Dispositivo
o imagen de fs
En el caso de un fs virtual
cualquier texto es válido

Lugar de montaje



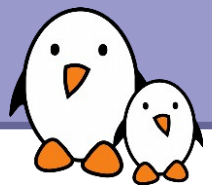
Detalles de /proc

Algunos ejemplos:

- `/proc/cpuinfo`: información del procesador
- `/proc/meminfo`: estado de la memoria
- `/proc/version`: información de compilación y versión del kernel
- `/proc/cmdline`: línea de comando del Kernel
- `/proc/<pid>/environ`: entorno de ejecución
- `/proc/<pid>/cmdline`: linea de comando del proceso

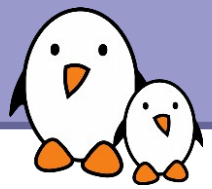
... y mucho más! Véalo con sus propios ojos!

Muchos detalles acerca de `/proc` disponibles en:
`Documentation/filesystems/proc.txt`
(casi 2000 líneas) en las fuentes del Kernel.



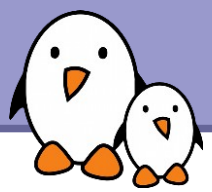
Kernel Linux

Esquema de versiones y modelo de desarrollo

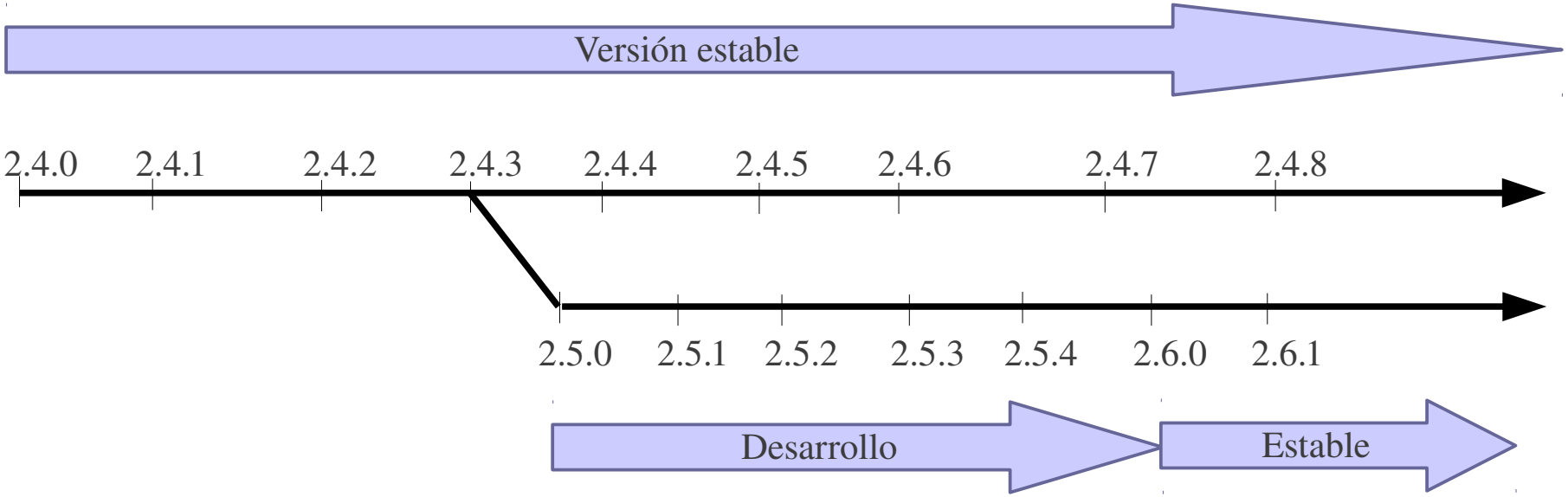


Hasta 2.6

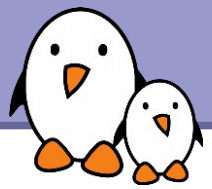
- Una nueva rama estable cada 2 o 3 años
  - Identificada por un número par en el medio
  - Ejemplos: `1.0, 2.0, 2.2, 2.4`
- Una rama de desarrollo para integrar nueva funcionalidad y cambios mayores
  - Identificada por un número impar en el medio
  - Ejemplos: `2.1, 2.3, 2.5`
  - Después de un tiempo, la rama de desarrollo pasaba a ser la base de una rama estable.
- Versiones menores de vez en cuando: `2.2.23, 2.5.12`, etc.



Hasta 2.6

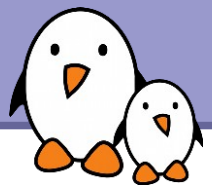


Nota: En realidad existen muchas más versiones menores en cada rama.



Cambios desde Linux 2.6

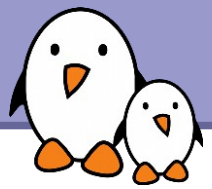
- Desde `2.6.0`, los programadores pudieron agregar muchas nuevas funciones de a una a paso constante sin tener que hacer grandes cambios en los subsistemas existentes.
- Abrir una nueva rama de desarrollo Linux `2.7` (o `2.9`) va a ser necesario cuando Linux `2.6` no pueda acomodar más funciones importantes sin cambios muy grandes.
- Gracias a esto, muchas nuevas funciones llegan a los usuarios a un paso mucho más rápido.



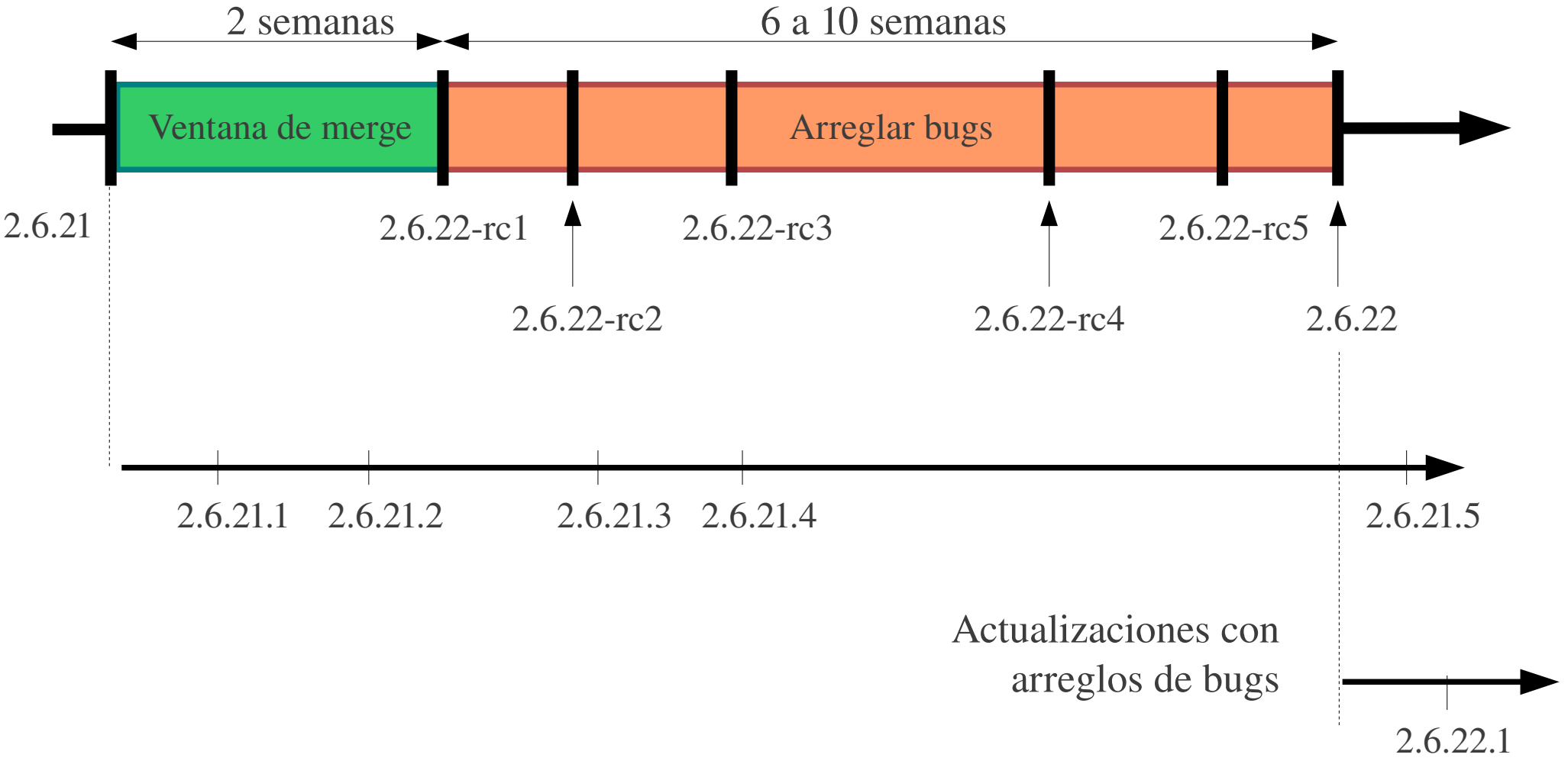
Cambios desde Linux 2.6

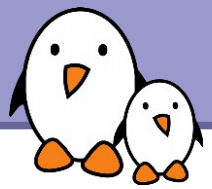
Desde 2.6.14, los programadores del Kernel se pusieron de acuerdo en el siguiente modelo de desarrollo:

- Después de un release de una versión `2.6.x`, se abre una ventana de 2 semanas para merge durante la cual los principales cambios son aplicados.
- Se cierra la ventana de merge con el release de la versión `2.6.(x+1)-rc1`
- Se abre el período para arreglar bugs, de 6 a 10 semanas.
- En intervalos regulares durante el período de arreglo de bugs se liberan las versiones de prueba `2.6.(x+1)-rcY`.
- Cuando se lo considera suficientemente estable, el Kernel `2.6.(x+1)` es liberado, y el proceso vuelve a comenzar.



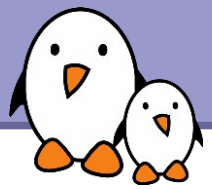
Ventanas de merge y arreglos





Más estabilidad con kernel 2.6

- Problema: los arreglos de bugs y seguridad se hacen sólo en la versión más nueva (a lo sumo dos anteriores). Las distribuciones, por supuesto, las aplican en los kernels que están usando.
- Algunas personas necesitan tener un Kernel nuevo pero con soporte por un plazo prolongado.
- Se puede tener soporte de largo plazo de los proveedores de Linux embebido.
- Se puede reutilizar las fuentes del Kernel de las versiones de Ubuntu LTS, o Debian (5 o 3 años de updates gratis).
- Algunas versiones se mantienen por un tiempo prolongado por kernel.org (Linux 2.6.27 por ejemplo).

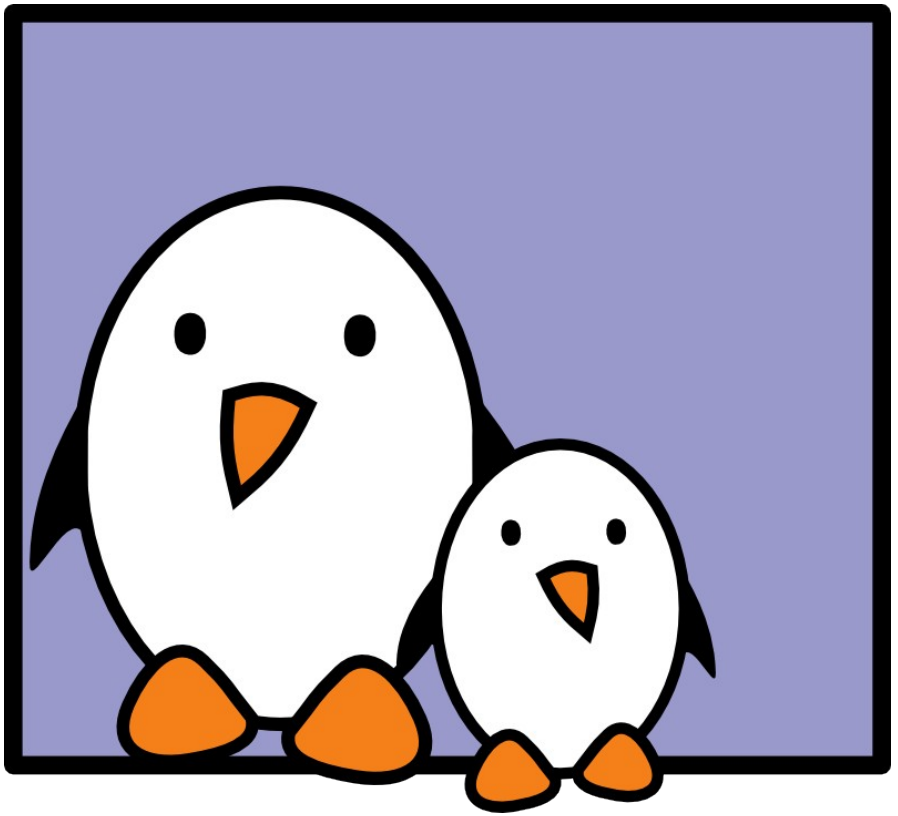


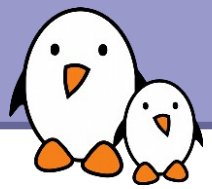
¿Qué tiene de nuevo cada versión?

```
commit 3c92c2ba33cd7d666c5f83cc32aa590e794e91b0
Author: Andi Kleen <ak@suse.de>
Date:   Tue Oct 11 01:28:33 2005 +0200

    [PATCH] i386: Don't discard upper 32bits of HWCR on K8

    Need to use long long, not long when RMWing a MSR. I think
    it's harmless right now, but still should be better fixed
    if AMD adds any bits in the upper 32bit of HWCR.

    Bug was introduced with the TLB flush filter fix for i386

    Signed-off-by: Andi Kleen <ak@suse.de>
    Signed-off-by: Linus Torvalds <torvalds@osdl.org>
...
```

- ¡¡El changelog oficial de cada versión es simplemente una larga lista de parches individuales!!
- Muy difícil de saber cuáles son los cambios claves y tener una vista global de los cambios.
- Afortunadamente, un resumen de los cambios fundamentales (con suficiente detalle) se encuentra en: http://wiki.kernelnewbies.org/LinuxChanges

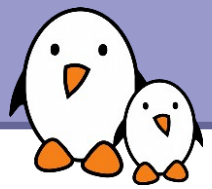
Cross-compiling toolchains



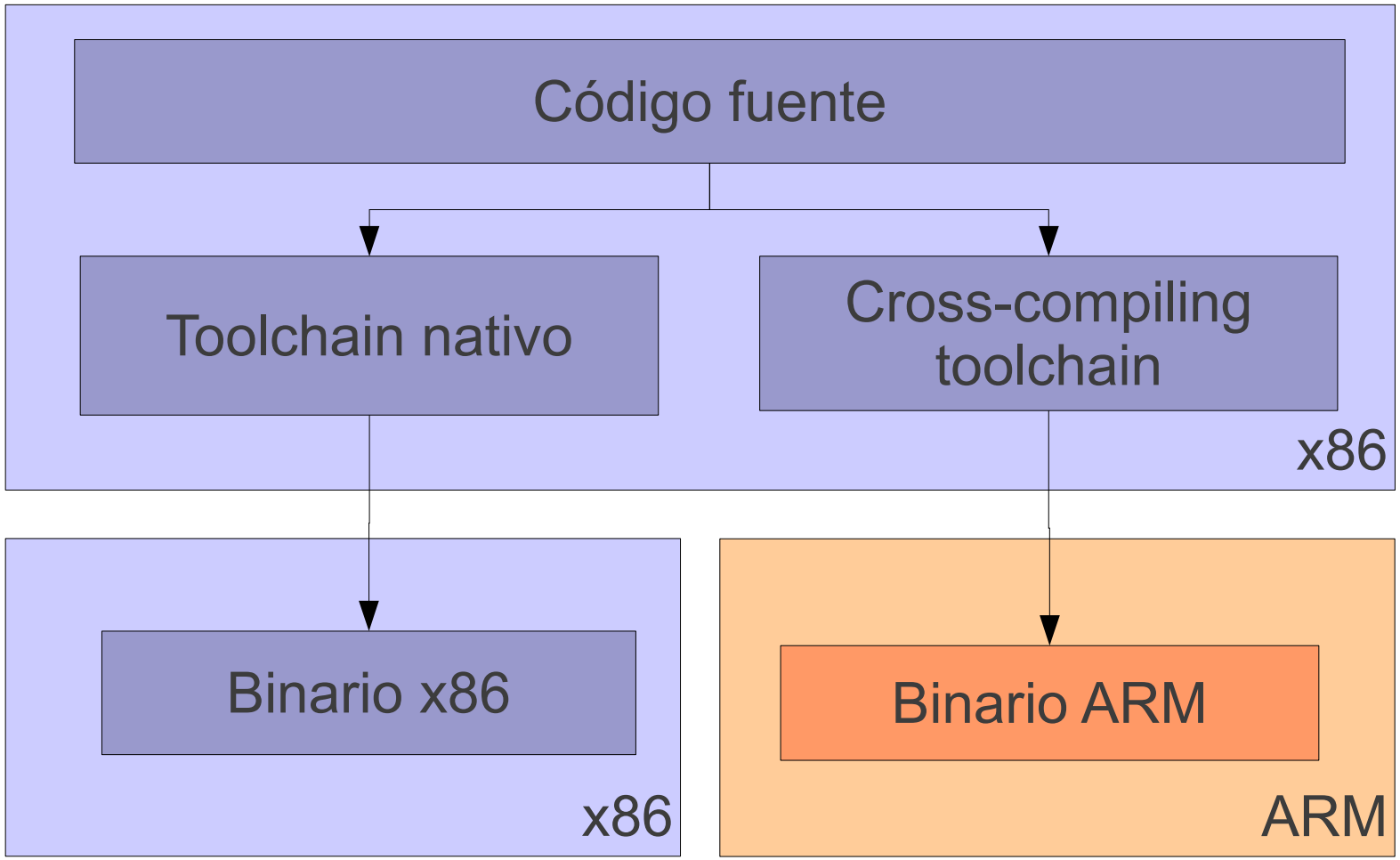


Definición

- Las herramientas de desarrollo que uno tiene normalmente en un desktop se llaman **toolchain nativo.**
- Este toolchain corre en el desktop y genera código para el desktop. Usualmente x86.
- Para sistemas embebidos usualmente no se puede o no es deseable.
  - El target tiene recursos limitados (poca RAM y almacenamiento)
  - El target es mucho más lento que el desktop.
  - Uno puede no querer tener instalados las herramientas de desarrollo en el target.
- Por lo tanto, generalmente se usan **cross-compiling toolchains.** Corren en el desktop y general código para el target.

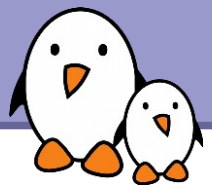


Definición



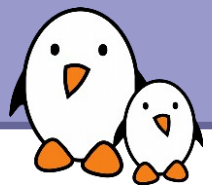
Máquina que compila

Máquina que ejecuta



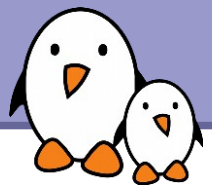
Componentes

- Binutils
- Kernel headers
- Librerías de C/C++
- Compilador GCC
- GDB debugger (opcional)



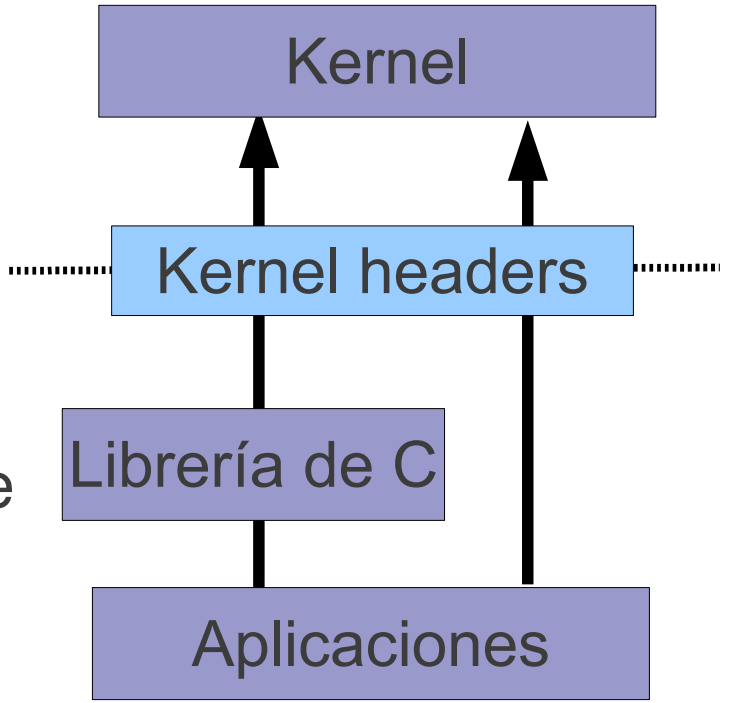
Binutils

- **Binutils** es un conjunto de herramientas para manipular binarios de una determinada arquitectura:
  - `as`, el ensamblador, genera código binario a partir de código assembler
  - `ld`, el linker
  - `ar, ranlib`, genera archivos `.a`, usados para librerías.
  - `objdump, readelf, size, nm, strings`, para inspeccionar binarios. Herramientas de análisis muy útiles!
  - `strip`, remueve partes de los binarios que no se usan.
    http://www.gnu.org/software/binutils/
- Licencia GPL

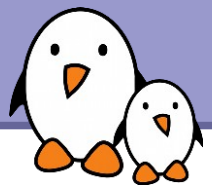


Kernel headers

- La librería de C y los programas tienen que interactuar con el Kernel
  - Llamadas a sistema disponibles.
  - Definiciones de constantes.
  - Estructura de datos, etc.
- Por lo tanto, para compilar la librería de C se necesitan los Kernel headers, y muchas aplicaciones también.

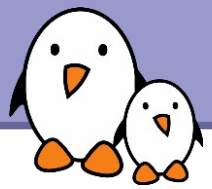


- Disponibles dentro de las fuentes del Kernel



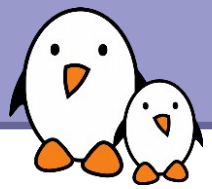
Kernel headers

- La ABI kernel-to-userspace es compatible hacia atrás:
  - Binarios generados con un toolchain usando headers anteriores que el Kernel que se está usando, van a funcionar pero no van a poder usar las prestaciones nuevas.
  - Binarios compilados contra headers más nuevos que el Kernel en uso pueden romperse si se usan características no presentes en el Kernel viejo.
  - No es necesario usar los headers más nuevos, salvo que se necesiten las últimas características.
- Los headers se extraen de las fuentes del Kernel con el comando `make headers_install`.



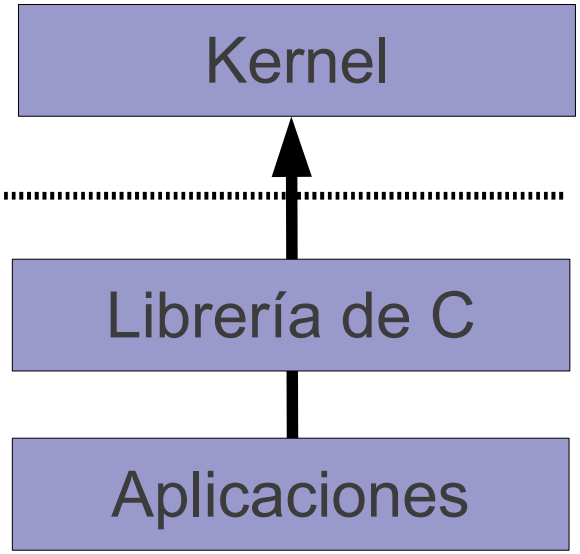
Compilador GCC

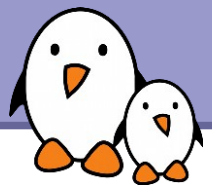
- Famoso compilador libre GCC
- Puede compilar C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, y generar código para muchas arquitecturas, incluyendo ARM, AVR, AVR32, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc.
- http://gcc.gnu.org/
- Disponible bajo licencia GPL.
  Las librerías que utiliza son LGPL.



Librería de C

- La librería de C es un componente esencial de una instalación de Linux.
  - Interfaz entre las aplicaciones y el Kernel.
  - Provee una API de C muy conocida para facilitar el desarrollo de aplicaciones.
- Existen muchas librerías de C: glibc, uClibc, eglibc, dietlibc, newlib, etc.
- La elección de una librería se debe hacer en el momento de generar el toolchain, ya que el GCC se compila contra la librería.

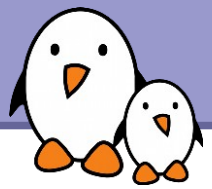




glibc

http://www.gnu.org/software/libc/

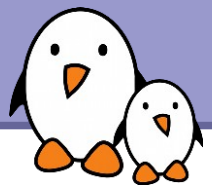
- Licencia: LGPL
- Librería de C del proyecto GNU.
- Diseñada para desempeño, respetar estándares y portabilidad.
- Encontrada en casi todos los sistemas GNU / Linux.
- Por supuesto, mantenida activamente.
- Bastante grande para sistemas embebidos: ~ 2.5 MB en `arm` (version 2.9 - `libc`: 1.5 MB, `libm`: 750 KB)



uClibc

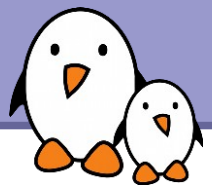
http://www.uclibc.org/ de CodePoet Consulting

- Licencia: LGPL
- Librería de C pequeña, para sistemas embebidos:
  - Muchas opciones de configuración: muchas funciones se pueden activar y desactivar usando la interfaz `menuconfig`.
  - Funciona en Linux/uClinux, en la mayoría de las plataformas embebidas.
  - No tiene una ABI estable, depende de la configuración de la librería.
  - Se ocupa del tamaño antes que del desempeño.
  - Se compila rápido.



uClibc

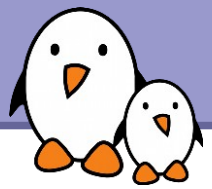
- La mayoría de las aplicaciones compilan contra uClibc, y todas las que se usan en embebidos.
- Tamaño (`arm`): 4 veces más chica que glibc!
  uClibc 0.9.30.1: ~ 600 KB (`libuClibc`: 460 KB, `libm`: 96KB)
  glibc 2.9: ~ 2.5 MB (para recordar)
- Usada en una gran cantidad de dispositivos embebidos, incluidos muchos de “consumer electronics”.
- Activamente soportada, gran base de usuarios.
- Ahora soportada por MontaVista, TimeSys y Wind River.



Querida encojí a los programas

Programa	*Usando librerías compartidas*		*Compilado estático*	
	glibc	***uClibc***	***glibc***	***uClibc***
Plain “hello world” (stripped)	5.6 K (glibc 2.9)	5.4 K (uClibc 0.9.30.1)	472 K (glibc 2.9)	18 K (uClibc 0.9.30.1)
Busybox (stripped)	245 K (older glibc)	231 K (older uClibc)	843 K (older glibc)	311 K (older uClibc)

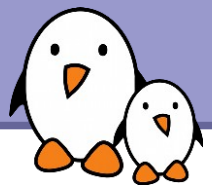
Comparación de tamaños de ejecutables en ARM



eglibc

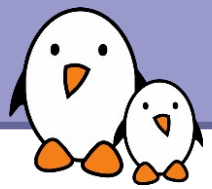
« Embedded glibc », LGPL

- Variante de la GNU C Library (GLIBC) diseñada para funcionar en sistemas embebidos.
- Compatible (fuente y binarios) con GLIBC
- Sus objetivos incluyen tamaño reducido, componentes configurables, mejor soporte para cross-compilación.
- Se puede compilar sin soporte par IPv6, localizaciones y muchas otras funciones.
- Soportada por un consorcio, con Freescale, MIPS, MontaVista y Wind River como miembros.
- Debian usa eglibc en todas las distribuciones.
- http://www.eglibc.org



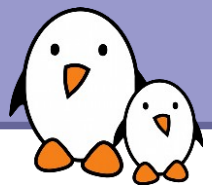
Otras librerías de C pequeñas

- Existen otras librerías de C más pequeñas, pero ninguna con el objetivo de compilar una gran cantidad de aplicaciones existentes.
- Requieren el código escrito especialmente para ellas.
- Opciones:
  - Dietlibc, http://www.fefe.de/dietlibc/. Aproximadamente 70 KB.
  - Newlib, http://sourceware.org/newlib/
  - Klibc, http://www.kernel.org/pub/linux/libs/klibc/, diseñada para ser usada durante el booteo.

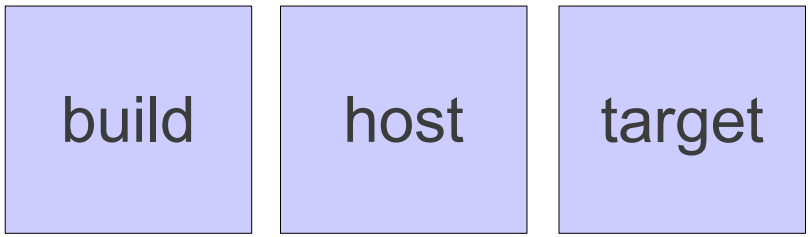


Compilando el toolchain

- Hay que diferenciar tres máquinas cuando se habla de la creación del toolchain:
  - La máquina de compilación, donde el toolchain se compila.
  - La máquina host, donde el toolchain se ejecuta.
  - La máquina destino, donde se van a ejecutar los binarios generados por el toolchain.
- Existen cuatro posibles maneras de crear el toolchain

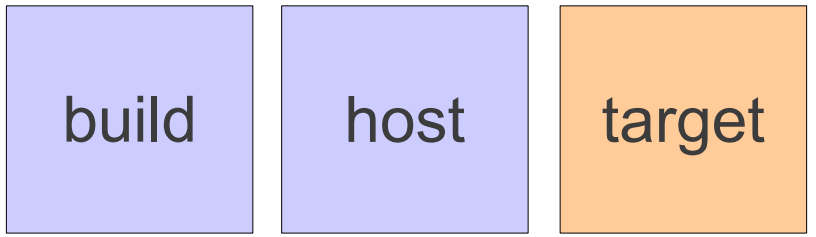


Compilando el toolchain



Nativo

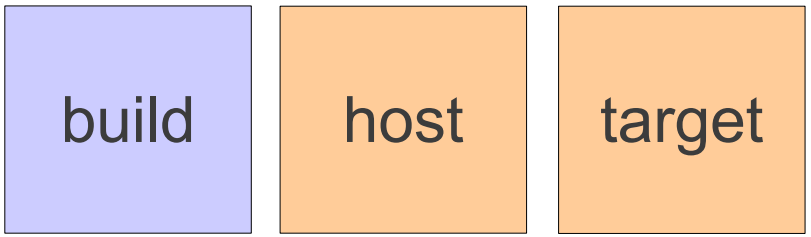
toolchain normal en un desktop



Cross

Genera un toolchain que corre en el desktop y crea código para ejecutarlo en el target

La solución más común en embebidos

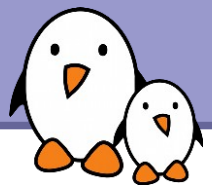


Cross-nativo

Genera un toolchain que corre en el target y genera código para el target

Canadian

Un toolchain que se compila en el desktop, se ejecuta en un server y genera código para el target



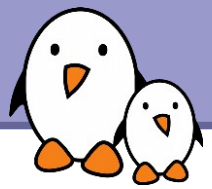
Compilando el toolchain

- Hay que tomar muchas decisiones al momento de hacer un toolchain.
  - Elegir la librería de C
  - Elegir la versión de los componentes que vamos a usar
- Y la configuración de todas estas cosas.
  - ¿Qué ABI debemos usar? Toolchains para ARM, pueden generar binarios usando la OABI (Old ABI) o la EABI (Embedded ABI), que son incompatibles.
  - ¿Debe el toolchain soportar punto flotante por software?, ¿o el hardware provee soporte para estas operaciones?
  - ¿Debe el toolchain soportar localización, IPv6, u otras características especiales?

Compilando el toolchain

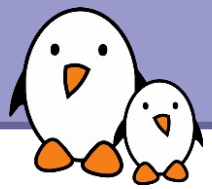
Reporte de compilación: http://kegel.com/crosstool/crosstool-0.43/buildlogs

gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	alpha
ICE gdb FAIL	ICE gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm9tdmi
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-iwmmxt
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-softfloat
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm-xscale
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armeb
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armv5b-softfloat
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	i686
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ia64
kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	m68k
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mips
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mipsel
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-405
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-750
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-860
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	powerpc-970
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	s390



Pasos básicos

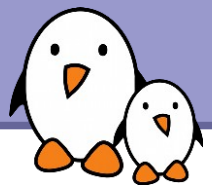
- Extraer e instalar los Kernel headers.
- Extraer, configurar, compilar e instalar binutils.
- Extraer, configurar y compilar la primera versión del GCC que genera binarios para el target. Lo vamos usar para compilar la librería de C de cross-compilación.
- Extraer, configurar y compilar la librería de C con el compilador generado en el paso anterior.
- Re configurar y compilar el gcc, contra la nueva librería de C. Así obtenemos el gcc que vamos a usar.



Toolchains caseros

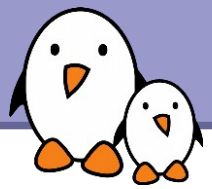
Compilar un cross toolchain a mano es una tarea dolorosa y difícil. Podemos tardar varios días en lograrlo!

- Muchos detalles que saber y entender. Muchos componentes que compilar.
- Muchas decisiones para tomar.
  (configuración de la librería de C para tu plataforma)
- Necesitamos fuentes de muchos programas diversos.
- Estar familiarizado con los problemas actuales del gcc y los parches para tu arquitectura.
- Es útil tener práctica compilando programas complejos.
- http://www.aleph1.co.uk/armlinux/docs/toolchain/toolchHOWTO.pdf
  ¡Te dice cuán entretenido puede ser!



Usar toolchains precompilados

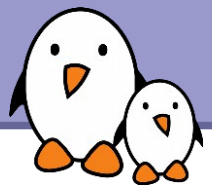
- La elección más popular, por ser lo más simple y fácil.
- Primero, hay que determinar qué toolchain se necesita. CPU, endianism, librería de C, componentes, ABI, punto flotante, etc..
- Existen muchos toolchains precompilados:
  - CodeSourcery, http://www.codesourcery.com, es una referencia en el área. No te proveen Linux+uClibc.
  - Linaro mantiene toolchains optimizados para ARM: https://wiki.linaro.org/WorkingGroups/ToolChain (Linaro contrató a CodeSourcery y agregó cosas)
  - Más información en http://elinux.org/Toolchains



Usando toolchains precompilados

- Seguir las instrucciones provistas por el fabricante.
- Usualmente es cuestión de descomprimir el archivo y dejarlo en el lugar correspondiente:
  - ¡Los toolchains no suelen ser reubicables!
    Los tenés que tener en el lugar donde fueron diseñados.
  - A partir de gcc 4.x esto no es necesario, pero suele ser lo más común y sencillo.
- Finalmente, agregar el toolchain a nuestro `PATH`:

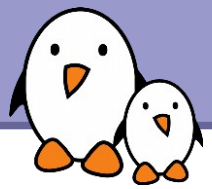
```
export PATH=/path/to/toolchain/bin/:$PATH
```



Herramientas para compilar

Otra solución es usar utilidades que automatizan el proceso de creación de un toolchain.

- Las mismas ventajas que los precompilados, no hay que meterse con la tediosa compilación manual.
- Ofrecen mayor flexibilidad que los toolchains ya compilados. Puedo seleccionar todas las opciones que quiero.
- Usualmente incluyen los parches recomendables para los distintos componentes en las distintas arquitecturas.
- Son scripts de shell o Makefiles que automatizan el proceso de descargar, extraer, configurar y compilar los distintos componentes.



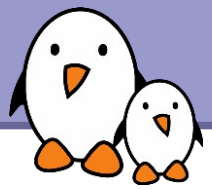
Herramientas para compilar

Crosstool

- Herramienta original, escrita por Dan Kegel
- Conjunto de scripts y parches, glibc-only
- No se mantiene más
- http://www.kegel.com/crosstool

Crosstool-ng

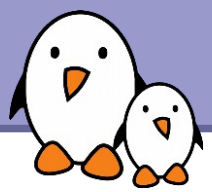
- Re-escritura de Crosstool, con una sistema de configuración del estilo de `menuconfig`.
- Más prestaciones: soporta uClibc, glibc, eglibc, hard y soft float, muchas arquitecturas.
- Activamente mantenido.
- http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool



Herramientas para compilar

Muchas herramientas para hacer rootfs también permiten compilar el toolchain.

- Buildroot
  - Basado en Makefile, sólo uClibc, mantenido por la comunidad. http://buildroot.uclibc.org
- PTXdist
  - Basado en Makefile, uClibc o glibc, mantenido principalmente por Pengutronix
  - http://www.pengutronix.de/software/ptxdist/index_en.html
- OpenEmbedded
  - El sistema de compilación más completo y complejo disponible.
  - http://www.openembedded.org/



Práctica – Usando CrosstoolNG

Momento de compilar su toolchain!

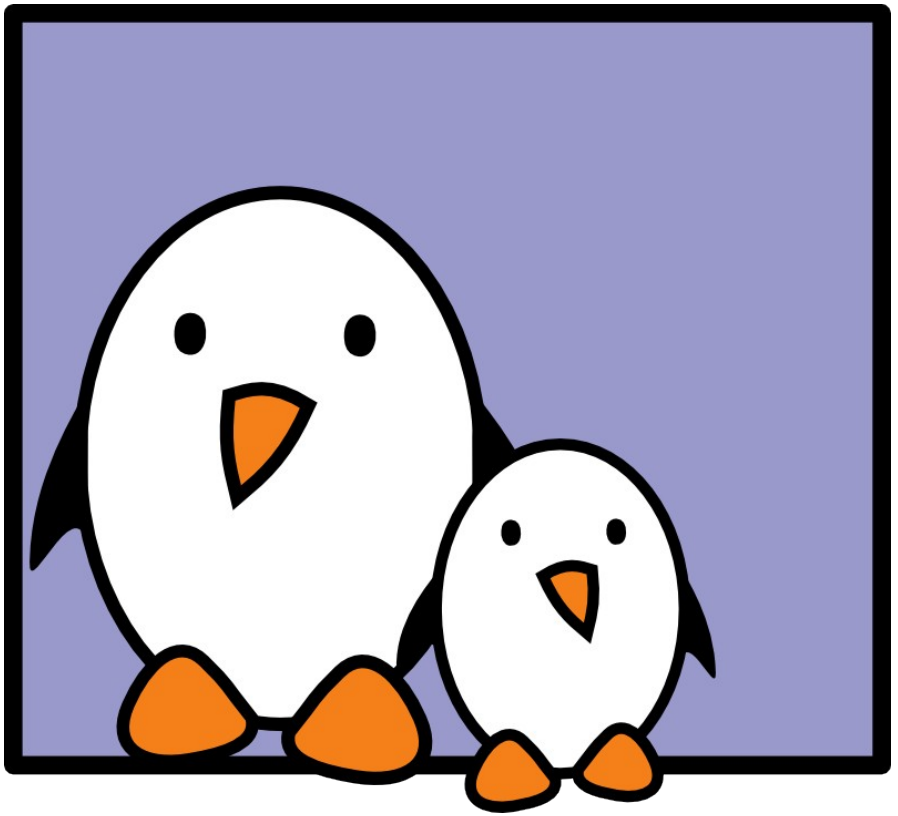
- Configurar Crosstool-NG
- Ejecutarlo para que compile su propio toolchain

- Mientras compila... ¿dudas?

SASE

Simposio Argentino de Sistemas Embebidos

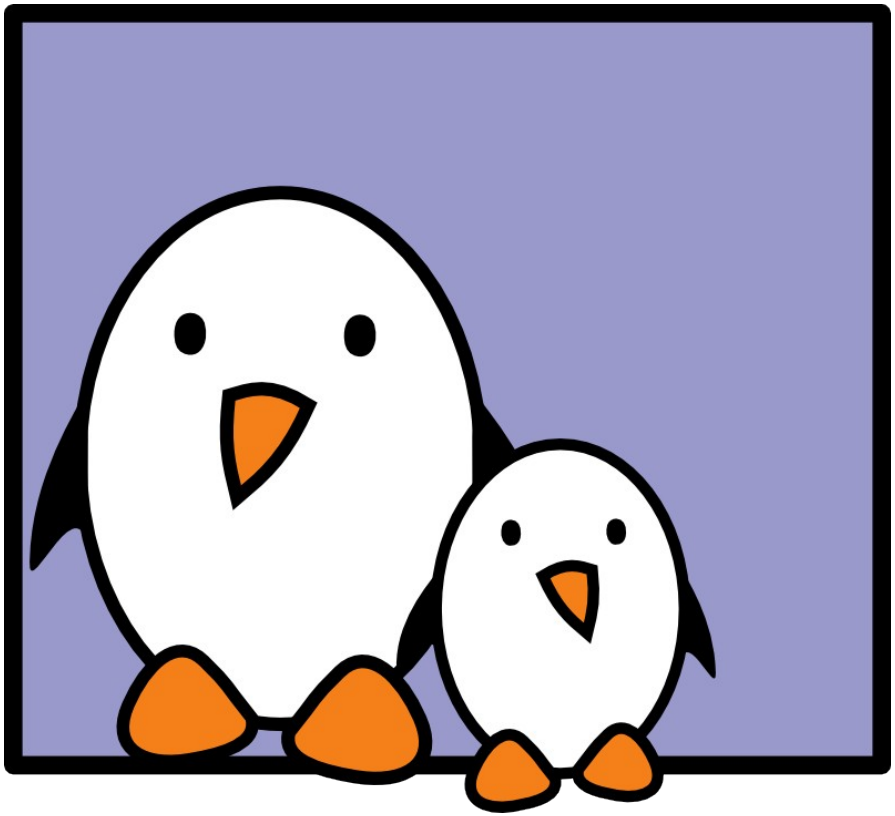
Workshop Linux Embebido

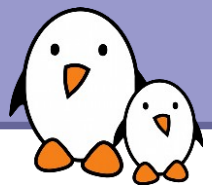


Lucas Chiesa
Joaquín de Andrés
Germán Bassi
Laboratorio
Sistemas embebidos
FIUBA

Basado en : http://free-electrons.com/docs/embedded-linux-intro

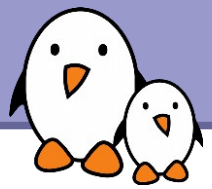
Bootloaders





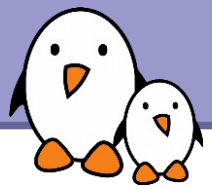
Bootloaders

- El bootloader es un programa responsable de:
  - Inicialización básica del hardware.
  - Cargar una aplicación binaria, usualmente el Kernel del SO, desde un almacenamiento flash, desde la red, o desde cualquier otro tipo de almacenamiento no volátil.
  - Descompresión eventual de la aplicación binaria.
  - Ejecución de la aplicación.
- Además de las funciones básicas, la mayoría de los bootloaders proveen un shell que implementan diferentes operaciones.
  - Inspección de memoria, diagnóstico de hardware, testeo, etc..



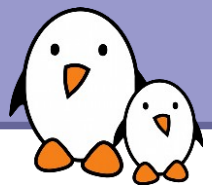
Bootloaders on x86

- Los procesadores x86 son usados típicamente en placas que contienen memoria no volátil donde se encuentra un programa, la BIOS.
- El programa es ejecutado desde la CPU después de un reset y es responsable de la inicialización básica del hardware y cargar una pequeña porción de código de almacenamiento no volátil.
  - Este código es usualmente los primeros 512 bytes de un disco rígido.
- Este código usualmente es la primera etapa del bootloader (1º stage bootloader), que va a cargar el bootloader “real”.
- Luego, el bootloader puede ofrecer todas sus características. Típicamente entienden sistemas de archivos, para cargar el Kernel directo del FS.



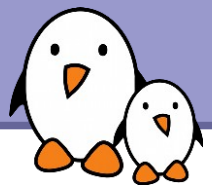
Bootloaders on x86

- GRUB, Grand Unified Bootloader, es el más poderoso.
  http://www.gnu.org/software/grub/
  - Puede leer muchos sistemas de archivos y cargar el Kernel y su configuración. Provee un shell muy poderoso con muchos comandos. Soporte de red, etc.
- LILO, el bootloader original de Linux
  http://freshmeat.net/projects/lilo/
- Syslinux, para bootear de red o de medios
  http://syslinux.zytor.com



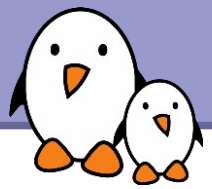
Bootloaders en embebidos

- En arquitecturas embebidas, el booteo de bajo nivel depende de cada CPU y cada placa.
  - Algunas placas tienen una NOR flash de la cual la CPU comienza la ejecución después del reset. En ese caso, el bootloader se debe grabar directamente en la dirección apropiada de esa NOR.
  - Algunas CPUs tiene código de booteo integrado en una ROM interna que automáticamente carga una porción de NAND flash en SRAM. En ese caso un bootloader mínimo de primera etapa se debe cargar en ese lugar de NAND. Luego, éste va a cargar el bootloader definitivo en DRAM y ejecutarlo. (BootROM on AT91SAM CPUs, Steppingstone on S3C24xx CPUs, etc.).
- El bootloader en embebidos inicia justo después de un reset de la CPU. Debe inicializar todos los dispositivos, incluyendo el controlador de memoria para poder acceder a la DRAM.
- Como el proceso de booteo es altamente dependiendo del fabricante es necesario tener la documentación propia del equipo.

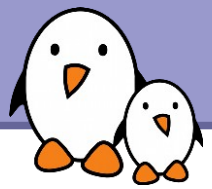


Bootloaders en embebidos

- Nos vamos a centrar en el bootloader principal, por ser el más genérico y por ofrecer la mayor cantidad de prestaciones.
- Hay varios bootloaders libres genéricos.
  Estos son los más populares:
  - U-Boot, el “universal bootloader” por Denx
    El más usado en ARM, también usado en PPC, MIPS, x86, m68k, NIOS, etc. El estándar de-facto actual, y es el que vamos a utilizar.
    http://www.denx.de/wiki/U-Boot
  - Barebox, un nuevo bootloader escrito como sucesor a U-Boot. Mejor diseño, mejor código y en desarrollo activo pero todavía no tiene tanto soporte de hardware como U-Boot.
    http://www.barebox.org
- Hay muchos otros bootloaders, libres y propietarios, usualmente específicos para alguna arquitectura.
  - RedBoot, Yaboot, PMON, etc.

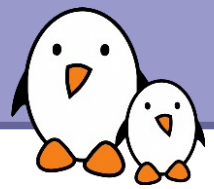


Accediendo a una consola serie

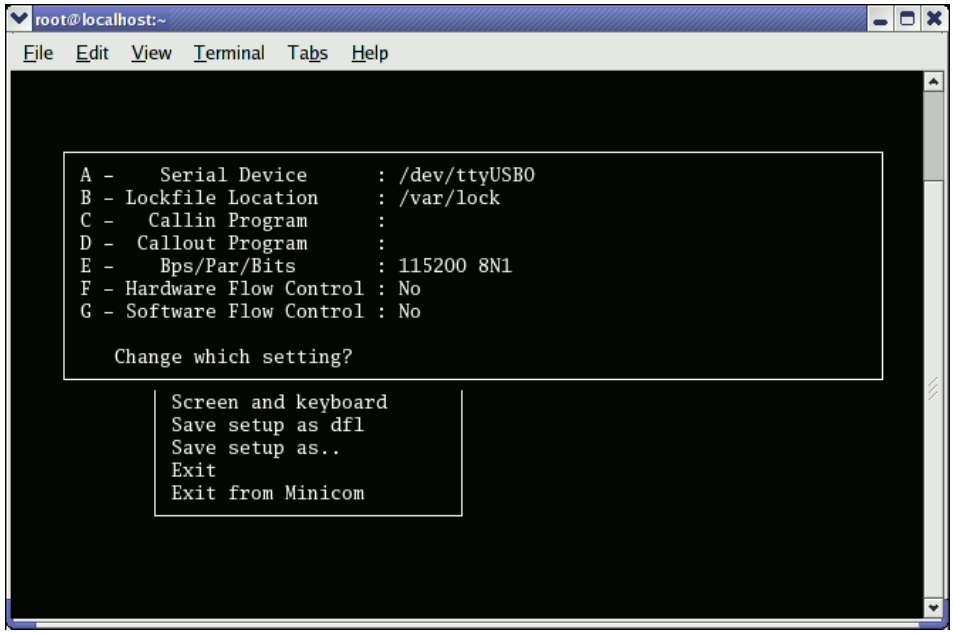


Minicom

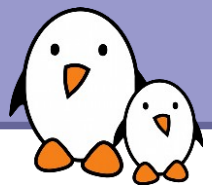
- Definición: programa de comunicación serie.
- Disponible en todas las distribuciones GNU / Linux.
- Capacidades:
  - Consola serie a un sistema Unix remoto
  - Transferencia de archivos
  - Controlar módems y dial-up
  - Configuración del puerto serie



Minicom

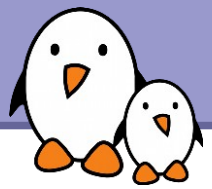


- Abrirlo ejecutando `minicom -s` para configurar minicom
- Un poco austero al principio, pero después resulta amistoso.



Otras terminales series

- GTKTerm: http://www.jls-info.com/julien/linux/
  Gráfico. Menos poderoso que Minicom, pero con una interfaz más simple y más atractiva. Disponible en distros recientes.
- CuteCom: http://cutecom.sourceforge.net/
  Otra terminal gráfica amistosa. Disponible en distros recientes.
- picocom: http://freshmeat.net/projects/picocom/
  Emulador muy pequeño (20K), se puede usar embebido
- GNU Screen: también se puede usar como terminal serie:
  `screen <device> <baudrate>`
  Ejemplo:
  `screen /dev/ttyS0 115200`

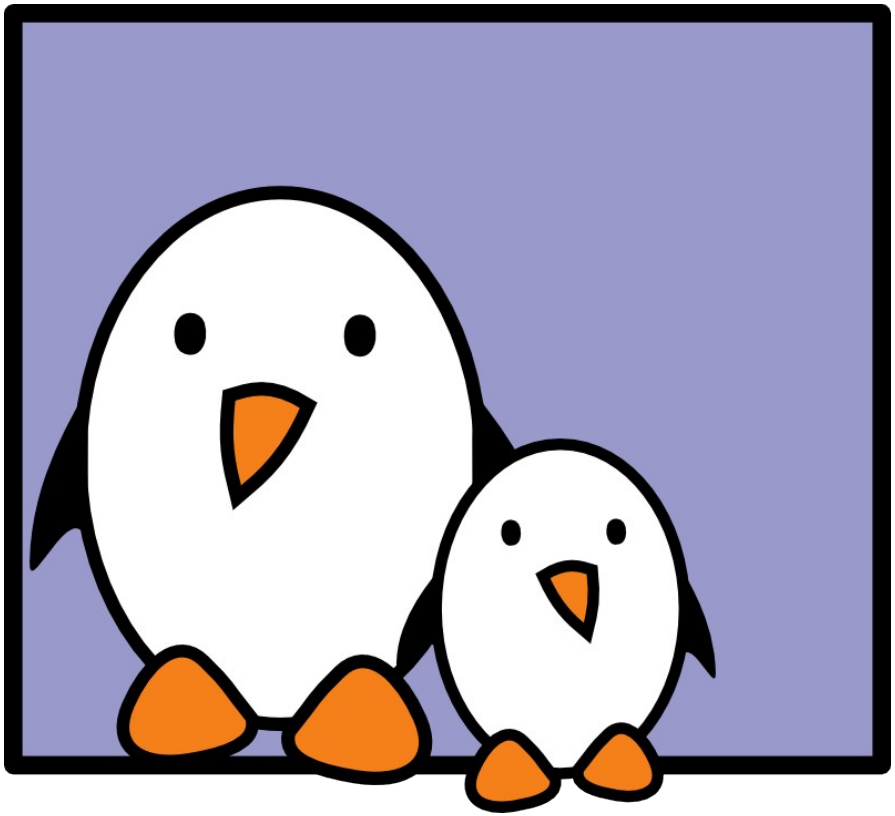


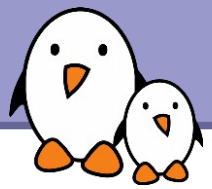
Práctica – Bootloaders

Momento de conectarse a U-Boot!

- Usar una terminal serie.
- Conocer los comandos de U-Boot

Uso embebido del Kernel Linux

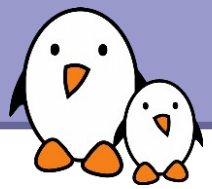




Contenidos

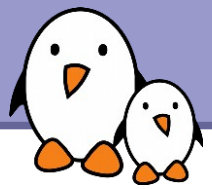
Compilando y booteando

- Fuentes del Kernel
- Configuración del Kernel
- Compilando el Kernel
- Inicio del sistema
- Archivos de dispositivos
- Cross compilando el Kernel



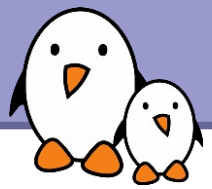
Compilando y Booteando el Kernel

Fuentes del Kernel Linux



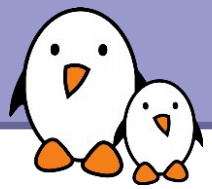
Fuentes del Kernel

- La versión oficial del Kernel, liberada por Linus Torvals, está disponible en http://www.kernel.org
  - Esta versión sigue el modelo de desarrollo descripto anteriormente.
  - Sin embargo, puede no contener los últimos cambios de un área específica. Algunos cambios pudieron darse después de la ventana de merge o estar todavía en desarrollo.
- Muchas sub-comunidades del Kernel mantienen sus propias versiones, usualmente con funciones más nuevas y menos estables.
  - Comunidades de arquitecturas (ARM, MIPS, PowerPC, etc.), comunidades de drivers (I2C, SPI, USB, PCI, red, etc.), otras comunidades (real-time, etc.)
  - Usualmente no liberan versiones oficiales sino que mantienen las ramas de desarrollo.

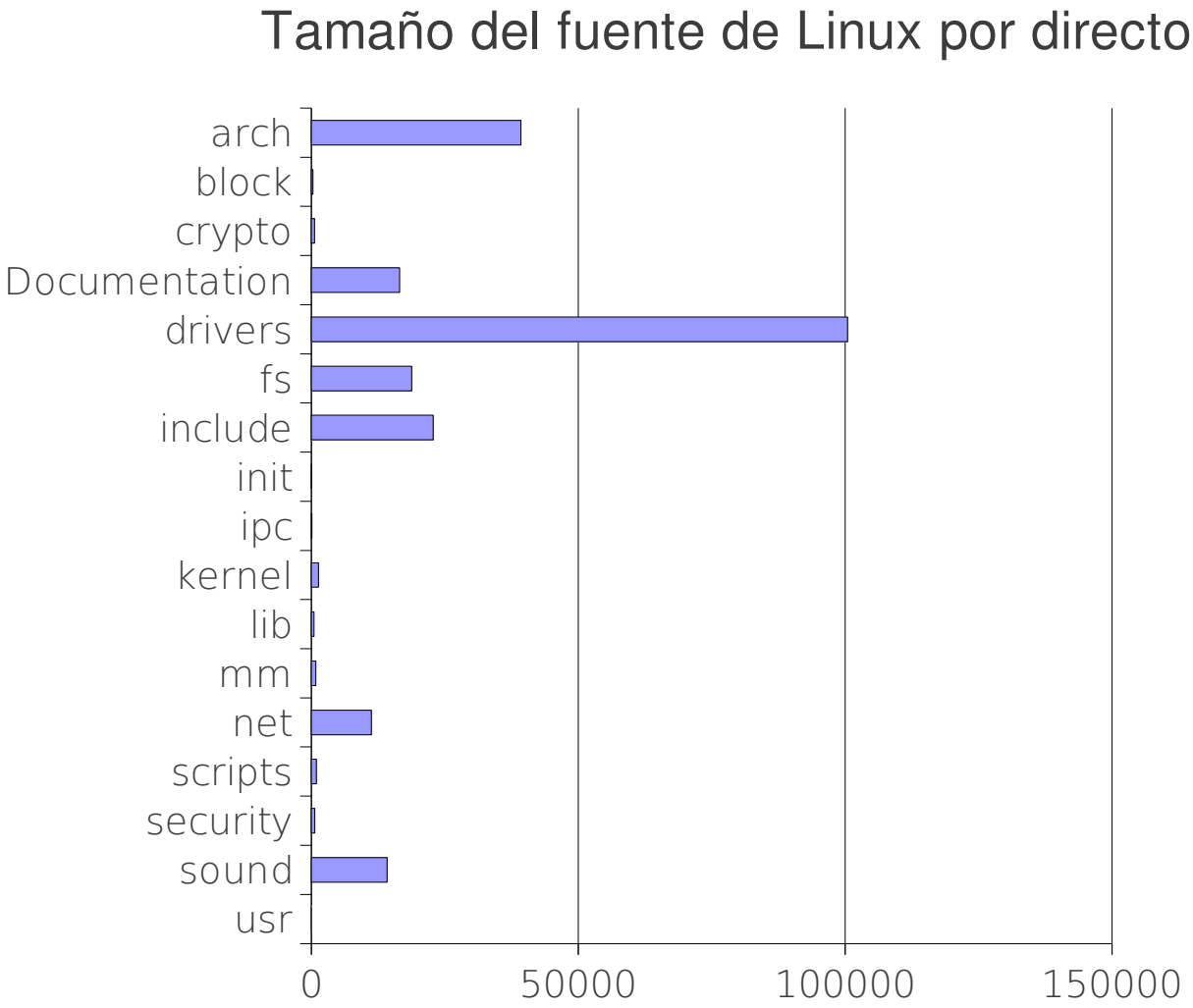


Tamaño del Kernel Linux

- Fuentes del Linux 2.6.31:
  Tamaño crudo: 350 MB (30,900 archivos, ~ 12,000,000 líneas)
  `gzip`: 75 MB
  `bzip2`: 59 MB (mejor)
  `lzma`: 49 MB (el mejor)
- Un Linux mínimo 2.6.29 compilado con la opción `CONFIG_EMBEDDED`, para bootear en un QEMU PC (disco IDE, ext2 filesystem, binarios ELF): 532 KB (comprimido), 1325 KB (descomprimido)
- ¿Por qué son tan grandes las fuentes?
  Proveen miles de drivers, muchos protocolos de red, muchas arquitecturas, muchos sistemas de archivos, etc...
- El núcleo de Linux (scheduler, manejador de memoria...) es bastante chico.



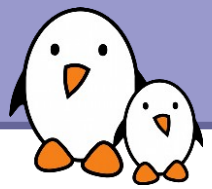
Tamaño del Kernel Linux



Linux 2.6.17

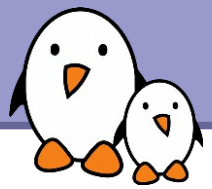
Medido con:

```
du -s --apparent-size
```



Bajando las fuentes

- Archivos completos
  - Contiene las fuentes completas del kernel.
  - Bastante para bajar y para descomprimir. Es necesario hacerlo por lo menos una vez.
  - Ejemplo: http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.7.tar.bz2
- Parches incrementales entre versiones
  - Asume que tenés una versión base y que aplicaste todos los parches en el orden correcto.
  - Rápido para bajar y aplicar
  - Ejemplo
    http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.bz2 (2.6.13 to 2.6.14)
    http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.7.bz2 (2.6.14 to 2.6.14.7)
- Todas las versiones anteriores están disponibles en: http://kernel.org/pub/linux/kernel/



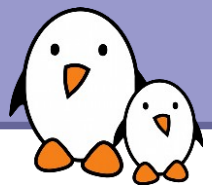
Aplicando un parche a Linux

- Siempre aplicar a la versión `x.y.<z-1>`
  Se pueden bajar comprimidos en
  `gzip, bzip2`.
- Siempre producidos para `n=1`
  Ejemplo:

```
cd linux-2.6.13
bzcat ../patch-2.6.14.bz2 | patch -p1
bzcat ../patch-2.6.14.7.bz2 | patch -p1
cd ..; mv linux-2.6.13 linux-2.6.14.7
```

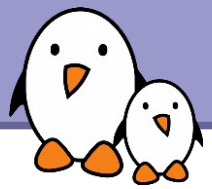
- Mantener el archivo comprimido: Sirve para verificar la firma.
  Uno puede ver y editar el archivo sin descomprimirlo usando `vim`:
  `vim patch-2.6.14.bz2` ((des)compresión on the fly)

Podés hacer `patch` 30% más rápido usando `-sp1` en lugar de `-p1` (**s**ilent)



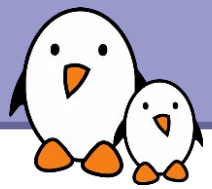
Práctica – Fuentes del Kernel

- Bajar las fuentes
- Descomprimirlas, familiarizarse.



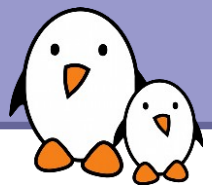
Compilando y booteando

Configuración del Kernel



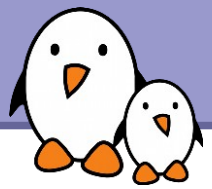
Configuración del Kernel

- Ya vimos que el Kernel contiene miles de drivers y otras cosas.
- Provee miles de opciones para compilar selectivamente diferentes partes del código.
- La configuración del Kernel es el proceso de selección de las opciones que queremos que tenga nuestro Kernel
- Este conjunto de opciones depende de
  - Nuestro hardware
  - De las prestaciones que necesitamos del Kernel



Configuración del Kernel

- La configuración se guarda en el archivo `.config` en la raíz de las fuentes
  - Archivo de texto plano, estilo `key=value`
- Como las opciones tienen dependencias es raro editar el archivo a mano. Se usan interfaces gráficas:
  - `make [xconfig|gconfig|menuconfig|oldconfig]`
  - Estos son targets del Makefile del Kernel. Correr `make help` nos da una lista de targets disponibles.
- Para cambiar el Kernel en una distro GNU/Linux:
  la configuración usada normalmente está en `/boot/`, junto con la imagen del Kernel: `/boot/config-2.6.17-11-generic`

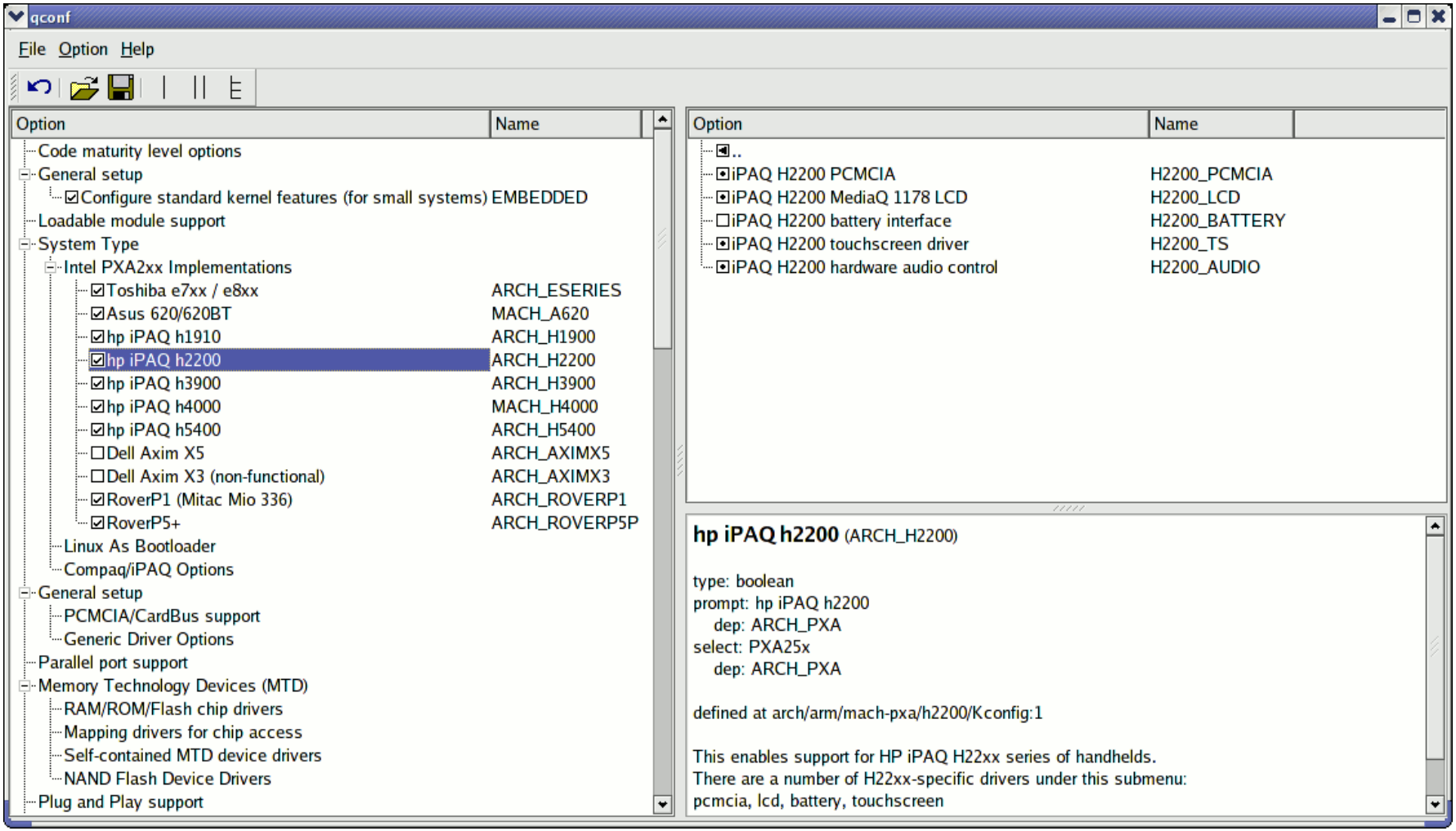


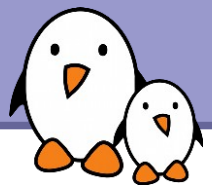
make xconfig

`make xconfig`

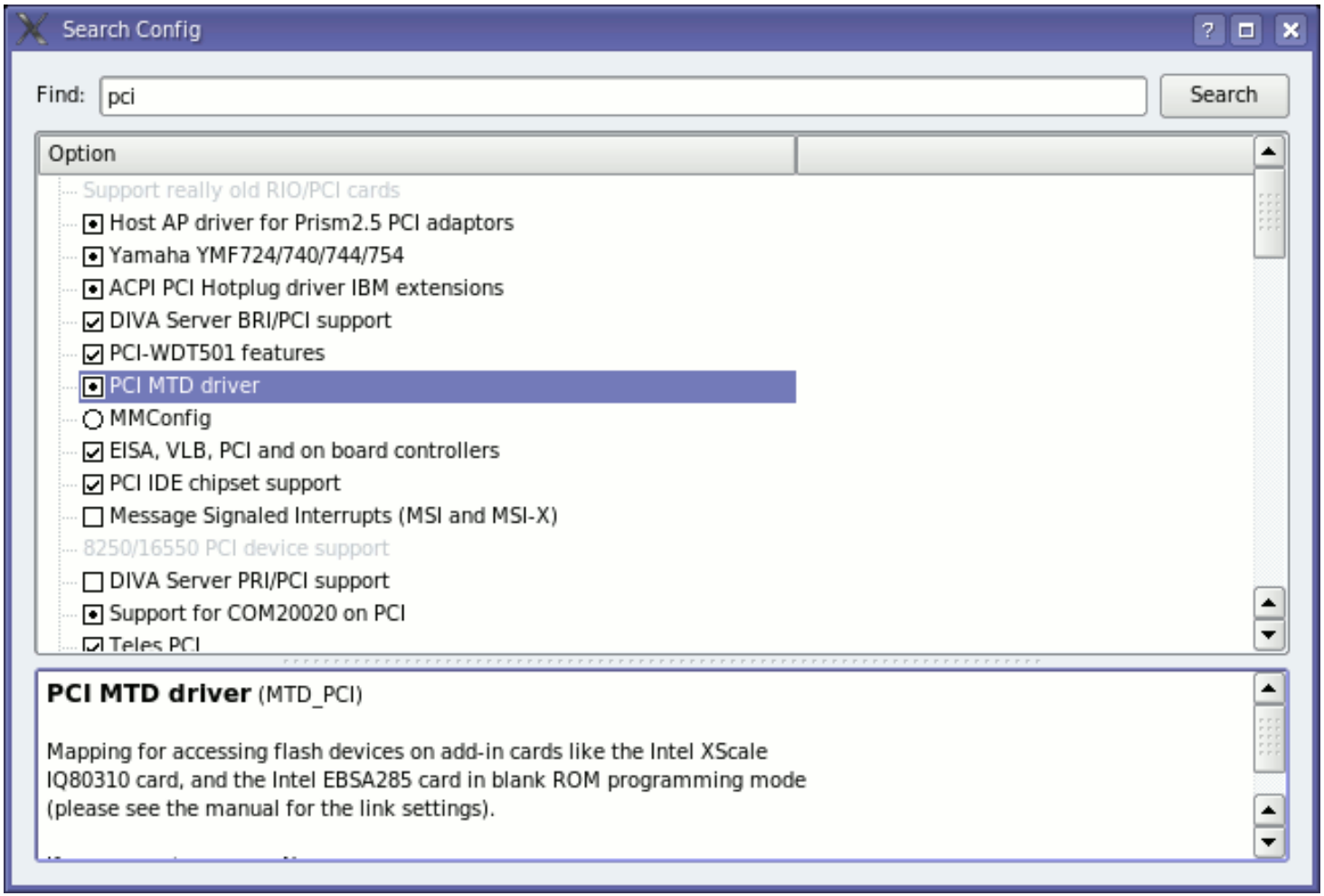
- La interfaz gráfica más común.
- Antes de usar leer:
  `help -> introduction: datos útiles`!
- File browser: facilita cargar archivos de configuración
- Interfaz de búsqueda de parámetros
- Paquetes de Debian / Ubuntu requeridos:
  `libqt3-mt-dev`, `g++`

make xconfig screenshot



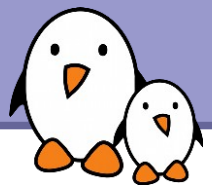


make xconfig: búsqueda



Busca una palabra cable en las descripciones

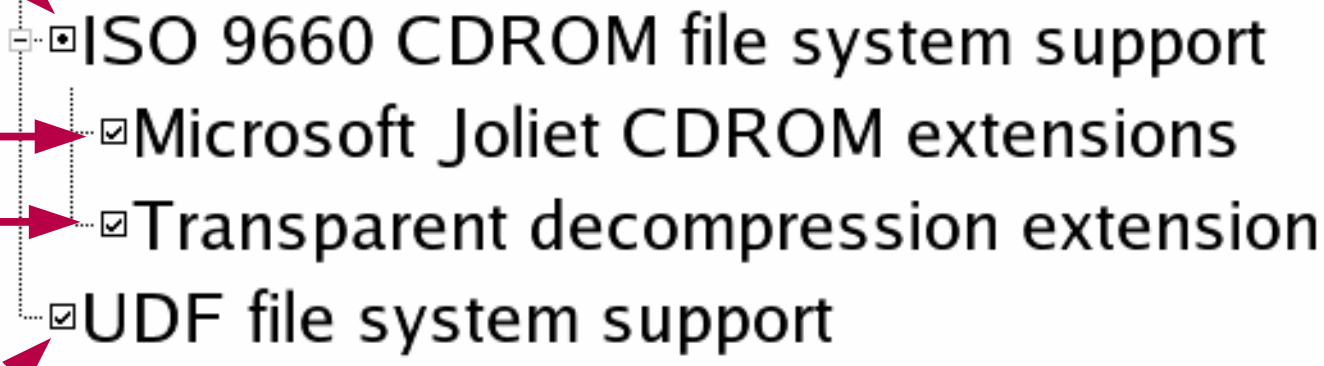
Te deja activar o no las opciones encontradas.



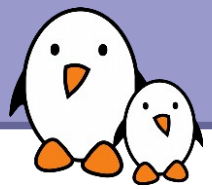
Opciones de configuración

Compilado como módulo (archivo separado)
`CONFIG_ISO9660_FS=m`

Opciones del driver
`CONFIG_JOLIET=y`
`CONFIG_ZISOFS=y`



Compilado estático en el Kernel
`CONFIG_UDF_FS=y`

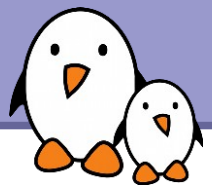


.config correspondiente

```
#
# CD-ROM/DVD Filesystems
#
CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_UDF_FS=y
CONFIG_UDF_NLS=y

#
# DOS/FAT/NT Filesystems
#
# CONFIG_MSDOS_FS is not set
# CONFIG_VFAT_FS is not set
CONFIG_NTFS_FS=m
# CONFIG_NTFS_DEBUG is not set
CONFIG_NTFS_RW=y
```

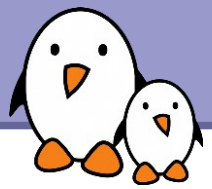
Nombre de la sección
(ayuda a encontrar settings)

Todos los parámetros comienzan con `CONFIG_`



Dependencias de las opciones

- Hay dependencias entre las opciones del Kernel
- Por ejemplo, activar un driver de red necesita tener el stack de red compilado.
- Dos tipos de dependencias:
  - Dependencias *"depends on"*. En este caso, la opción A que depende de B no es visible hasta que B no esté activada.
  - Dependencias *"select"*. En este caso, si trato de activar la opción A, entonces B es seleccionada automáticamente.
  - *make xconfig* permite ver todas las opciones, incluidas las que no se pueden seleccionar por faltar dependencias. En ese caso, las muestra en color gris.



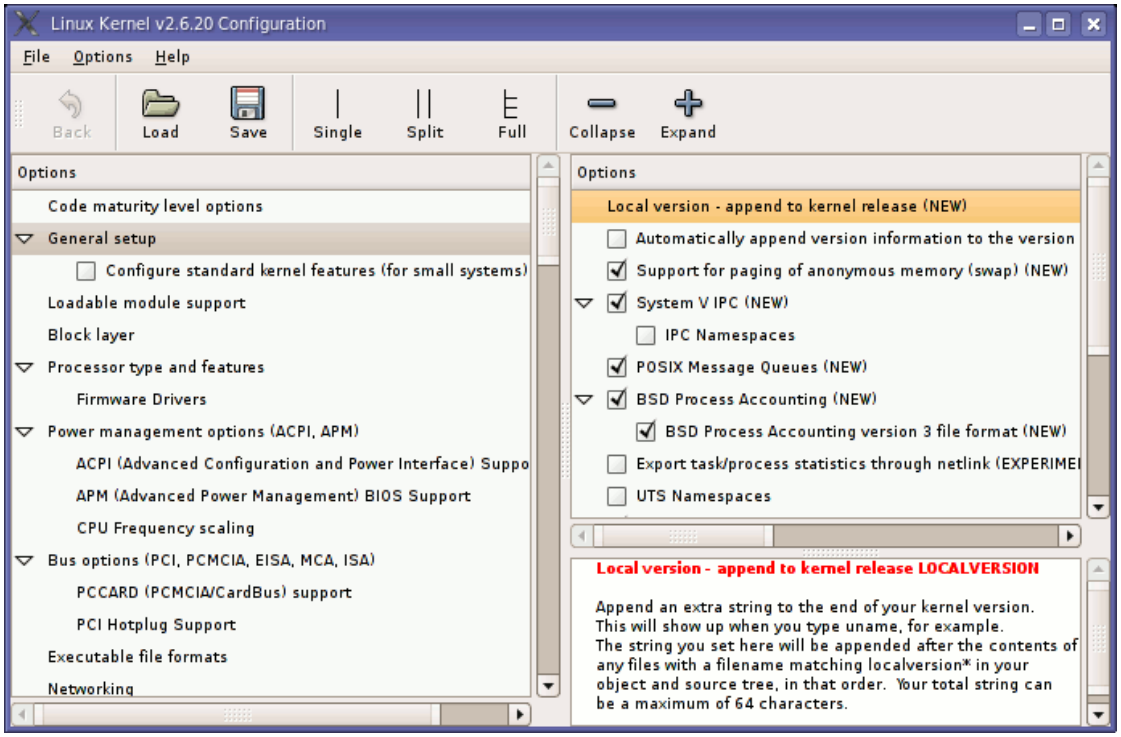
make gconfig

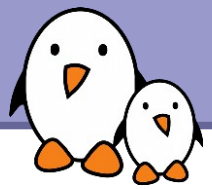
`make gconfig`

Nuevo *frontend* basado en GTK. Funcionalidad similar a la de `make xconfig`.

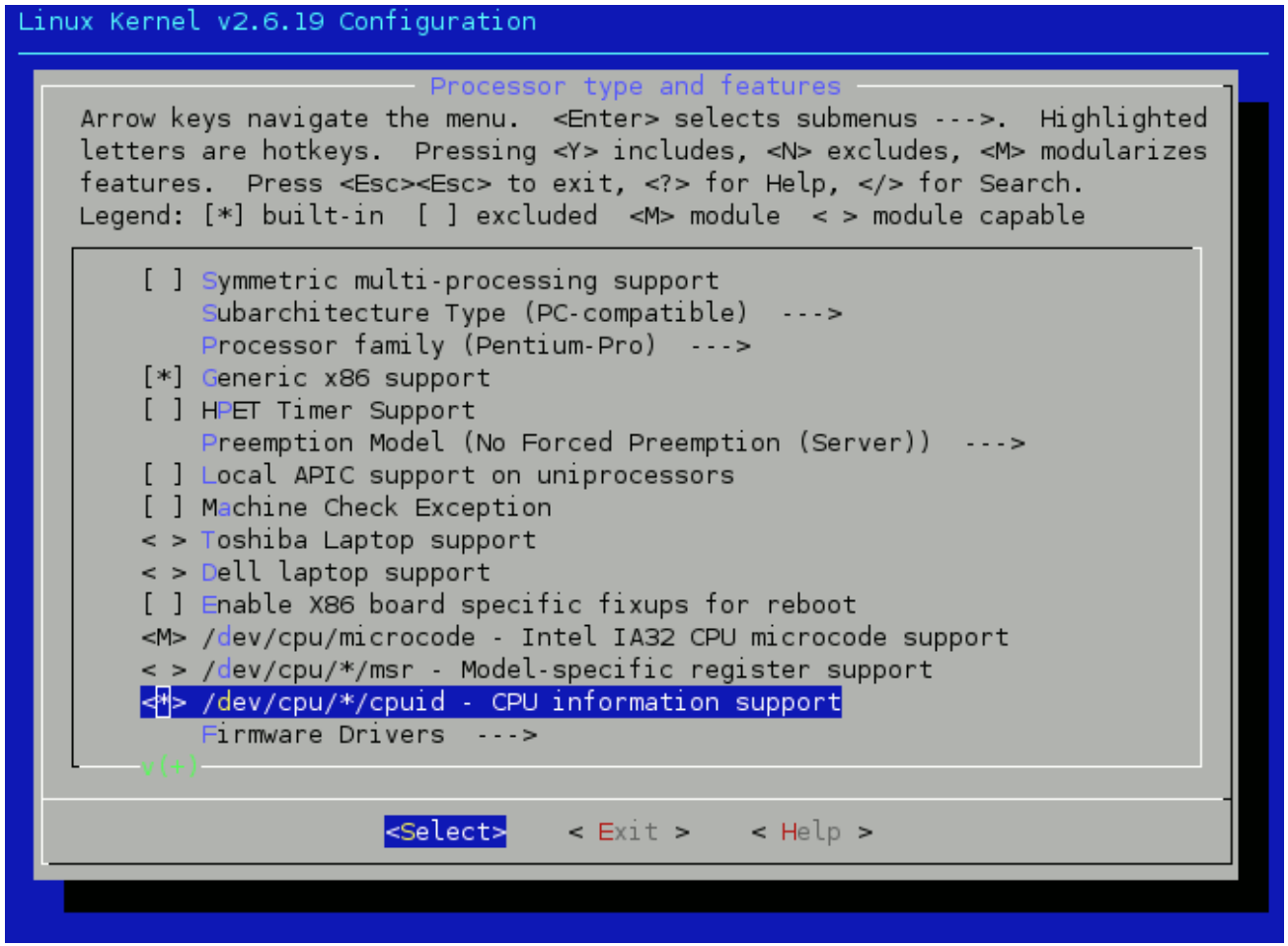
No se puede buscar.

Paquetes requeridos: `libglade2-dev`





make menuconfig

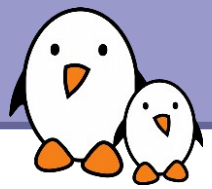


`make menuconfig`

Útil cuando no tengo interfaz gráfica. Muy cómoda también!

Misma interfaz que se usa en otras herramientas: BusyBox, buildroot...

Paquetes requeridos: `libncurses-dev`

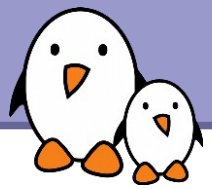


make oldconfig

`make oldconfig`

- ¡Frecuentemente necesario!
- Usado para actualizar un archivo `.config` de una versión anterior del Kernel
- Emite advertencias cuando se usaron configuraciones que no existen más en el nuevo Kernel
- Pregunta por el valor de nuevas opciones

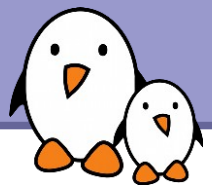
Si se edita a mano el `.config` es muy recomendable ejecutar `make oldconfig` después!



make allnoconfig

`make allnoconfig`

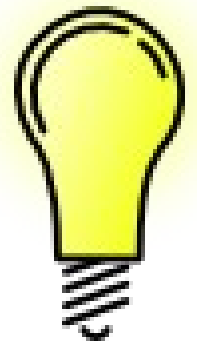
- Sólo activa las funciones fuertemente recomendadas
- Todo lo demás, no lo compila.
- Muy útil para partir de lo mínimo y agregar lo que necesitamos.
- ¡Más fácil agregar lo poco que necesitamos que sacar todo lo que sobra!

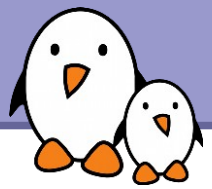


Deshaciendo cambios en la conf.

Un problema frecuente:

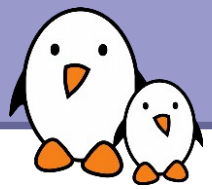
- Después de varios cambios de configuraciones el Kernel deja de funcionar.
- Si uno no se acuerda todos los cambios que fue haciendo, se puede recuperar la configuración anterior:
  `> cp .config.old .config`
- Todas las interfaces de configuración (`xconfig`, `menuconfig`, `allnoconfig`...) mantienen una copia de seguridad en `.config.old`.



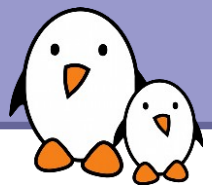


Configuraciones por arquitectura

- El conjunto de configuraciones depende de la arquitectura
  - Algunas configuraciones son muy específicas.
  - La mayoría de las opciones (opciones globales del Kernel, subsistema de red, filesystems) son visibles en todas las archs.
- Por omisión, el sistema de compilación asume una compilación nativa.
- La arquitectura no se especifica entre las opciones de compilación, sino en un nivel superior.
- Vamos a ver cómo seleccionar la arquitectura que queremos.

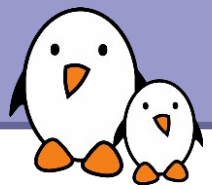


Compilando e instalando el Kernel
Para el sistema host



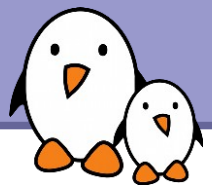
Compilación del kernel

- `make`
  - En el directorio principal del código
  - Recordar de usar `make -j n` si tenes más de un core n=cores+1
  - No tenés que ser root!
- Genera
  - `vmlinux`, imagen cruda, sin comprimir en formato ELF, útil para debugging pero no puede ser booteado.
  - `arch/<arch>/boot/*Image`, la imagen final, usualmente comprimida y puede ser booteada.
    - `bzImage` para x86, `zImage` para ARM, `vmImage.gz` para Blackfin, etc.
  - Los módulos de kernel están distribuidos por todas las fuentes y son archivos que terminan en `.ko`.



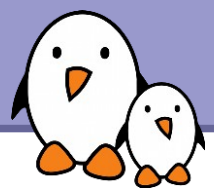
Instalación del Kernel

- `make install`
  - Instala el Kernel en el sistema host. Se debe correr como root.
- Instala
  - `/boot/vmlinuz-<version>`
    La imagen del Kernel comprimida. La misma que en `arch/<arch>/boot`
  - `/boot/System.map-<version>`
    Guarda la dirección de los símbolos del Kernel
  - `/boot/config-<version>`
    Configuración del Kernel para esa versión
  - Usualmente ejecuta la utilidad de configuración del bootloader para incorporar el nuevo Kernel



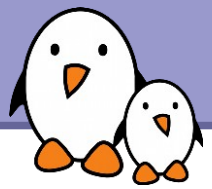
Instalación de los módulos

- `make modules_install`
  - Instala los módulos en el sistema host. Se debe ejecutar como root.
- Instala los módulos en /lib/modules/<version>/
  - `kernel/`
    Guarda todos archivos los `.ko` (Kernel Object), manteniendo la estructura de las fuentes.
  - `modules.alias`
    Archivo usado por las herramientas que cargan módulos. Por ejemplo:
    `alias sound-service-?-0 snd_mixer_oss`
  - `modules.dep`
    Dependencias de los módulos
  - `modules.symbols`
    Muestra a qué módulo corresponde cada símbolo.



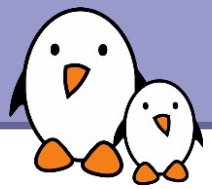
Limpiando la fuentes

- Limpiar los archivos generados por compilación (para forzar a compilar todo nuevamente):
  `make clean`
- Borra todos los archivos generados. Usado cuando cambiamos de arquitectura
  Cuidado: también borra el archivos `.config`!
  `make mrproper`
- También borrar los archivos de backup y archivos generados por patch. Útil cuando se está haciendo un parche.
  `make distclean`



Compilando y booteando Linux

Archivos de dispositivos

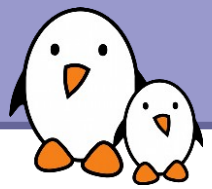


Dispositivos de caracteres

- Se acceden como un flujo secuencial de caracteres individuales
- Se identifican por el tipo `c` (`ls -l`):

```
crw-rw---- 1 root uucp   4,  64 Feb 23 2004 /dev/ttyS0
crw--w---- 1 jdoe tty  136,   1 Feb 23 2004 /dev/pts/1
crw------- 1 root root  13,  32 Feb 23 2004 /dev/input/mouse0
crw-rw-rw- 1 root root   1,   3 Feb 23 2004 /dev/null
```

- Ejemplos: teclados, mouse, puerto paralelo, IrDA, Bluetooth port, consolas, terminales, etc...

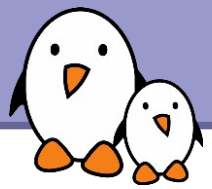


Dispositivos de bloques

- Se acceden por bloques de datos de algún tamaño fijo. Pueden ser leídos en cualquier orden.
- Se identifican por el tipo de archivo `b` (`ls -l`):

```
brw-rw----   1 root disk     3,   1 Feb 23  2004 hda1
brw-rw----   1 jdoe floppy   2,   0 Feb 23  2004 fd0
brw-rw----   1 root disk     7,   0 Feb 23  2004 loop0
brw-rw----   1 root disk     1,   1 Feb 23  2004 ram1
brw-------   1 root root     8,   1 Feb 23  2004 sda1
```

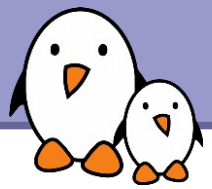
- Ejemplos: discos rígidos y floppy, discos ram, dispositivos loop...



Números minor y major

Como se ve en los ejemplos anteriores los archivos tienen dos números asociados:

- El primero: el número *major*
- El segundo: el número *minor*
- Los números major y minor son números usados por el Kernel para vincular un driver con un dispositivo. Los nombres de los archivos no le importan al Kernel!
- Para saber a qué driver corresponde cada número, ver `Documentation/devices.txt`.



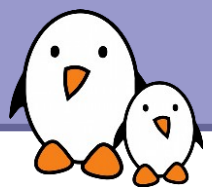
Creación de los archivos

- No se crean cuando se carga el driver
- Deben ser creados de antemano:

```
sudo mknod /dev/<device> [c|b] <major> <minor>
```

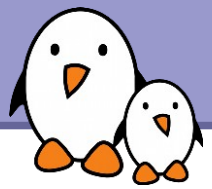
- Ejemplos:

```
sudo mknod /dev/ttyS0 c 4 64
sudo mknod /dev/hda1 b 3 1
```



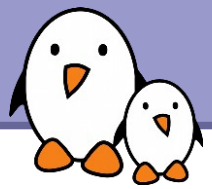
Práctica – Configurar y compilar

- Configurar su Kernel
- Compilarlo
- Bootearlo en una máquina virtual



Compilando y booteando Linux

Inicio general del sistema

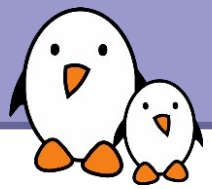


Secuencia de booteo tradicional

Bootloader
- Ejecutado por el hardware desde una posición fija de ROM / Flash
- Inicializa el soporte de los dispositivos donde se encuentra el kernel. (almacenamiento, red, medios removibles)
- Carga el Kernel en RAM
- Ejecuta la imagen del Kernel (con unos parámetros determinados)

Kernel
- Se auto descomprime
- Inicializa el núcleo del Kernel y drivers compilados estáticos (necesarios para acceder al root filesystem)
- Monta el root filesystem (especificado por el parámetro root)
- Ejecuta el primer programa de userspace (especificado por el parámetro init)

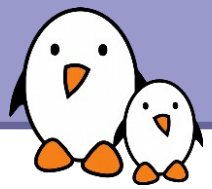
Primer programa de userspace
- Configura userspace y continúa levantando servicios



Parámetros del Kernel

El Kernel acepta parámetros al momento de bootear

- Son parte de la configuración del bootloader
- Son copiados a RAM por el bootloader a la posición de RAM donde los espera el Kernel.
- Útiles para modificar el comportamiento del Kernel al momento de booteo, sin recompilarlo.
- Útiles para hacer inicialización avanzada de drivers y el Kernel sin la necesidad de tener complejos scripts de userspace.



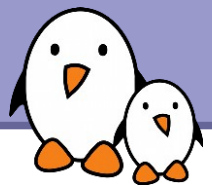
Ejemplo de los parámetros

HP iPAQ h2200 PDA ejemplo:

```
root=/dev/ram0 \
rw \
init=/linuxrc \
console=ttyS0,115200n8 \
console=tty0 \
ramdisk_size=8192 \
cachepolicy=writethrough
```

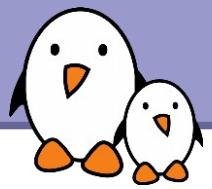
Root fs (primer ramdisk)
Modo de montaje del root fs
Primer programa userspace
Consola (serie)
Otra consola (framebuffer)
Otros parámetros...

Cientos de parámetros descriptos en:
`Documentation/kernel-parameters.txt`



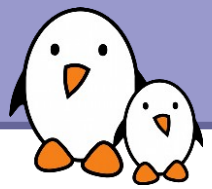
Desventajas

- Se asume que todos los drivers necesarios para montar el root fs (almacenamiento y fs) están compilados estáticos en el kernel.
- Se puede asumir en la mayoría de los sistemas embebidos que esto va a pasar, donde el hardware es conocido y el kernel está afinado al sistema.
- Es usualmente errónea en los sistemas desktop y servers donde una imagen tiene que levantar en una cantidad de dispositivos muy grande.
  - Se necesita más flexibilidad
  - Los módulos te dan esta flexibilidad, pero no están disponibles hasta montar el root filesystem.
  - Necesidad de soportar sistemas complejos (RAID, NFS, etc.)



Solución

- Una solución es incluir un pequeño rootfs temporal con los módulos necesarios dentro del Kernel mismo. Este sistema de archivos se llama initramfs.
- El initramfs es un sistema de archivos comprimido cpio con un rootfs muy básico.
  - Un cpio es como un zip pero con un formato mucho más simple.
- Los scripts del initramfs van a detectar el hardware, cargar los módulos necesarios y montar el rootfs real.
- Finalmente, el los scripts del initramfs van a correr la aplicación init del rootfs real y continuar el booteo usual.
- Esta técnica reemplaza completamente los init ramdisks (initrds), que se usaban con Linux 2.4, pero no son más necesarios.



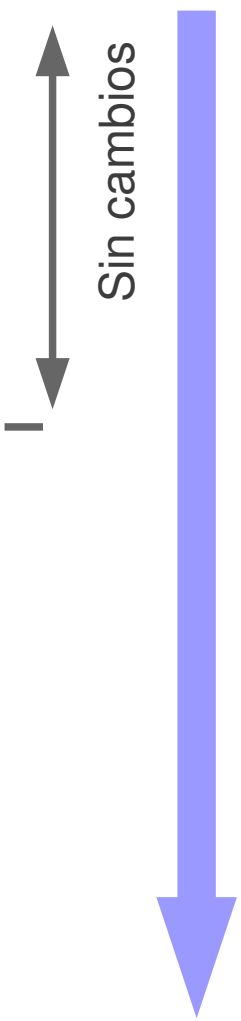
Secuencia de booteo con initramfs

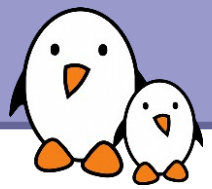
Bootloader
- Ejecutado por el hardware desde una posición fija de ROM / Flash
- Inicializa el soporte de los dispositivos donde se encuentra el kernel. (almacenamiento, red, medios removibles)
- Carga el Kernel en RAM
- Ejecuta la imagen del kernel (con unos parámetros determinados

Kernel
- Se descomprime
- Inicializa el núcleo y los drivers compilados estáticos.
- Descomprime el archivo cpio del initramfs. (si existe, dentro del kernel, sino el bootloader lo copia a una parte específica de memoria) y lo extrae en el cache del kernel.
- Si encontró un initramfs, ejecuta el primer programa dentro del initramfs. `/init`

Userspace: script `/init` (es lo mismo que un userspace normal)
- Corre comandos para configurar el dispositivo (configuración de red, montar `/proc` y `/sys`...)
- Monta el rootfs. Cambiar a ese filesystem (`switch_root`)
- Corre `/sbin/init`

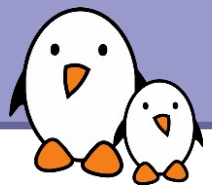
Userspace: `/sbin/init`
- Configura lo que no esté configurado por el initramfs.
- Levanta los servicios (demonios, servers) y programas de usuario





Ventajas initramfs

- Es una solución simple, meter un rootfs directo en el kernel o copiado por el bootloader.
- Es simplemente un archivo cpio comprimido que se extrae al cache de archivos. No necesita un bloque ni un driver de filesystem.
- Es más montar filesystem complejos desde scripts de userspace que desde código de kernel. Mueve la complejidad a espacio de usuario.
- Posibilita agregar archivos no GPL (firmware, drivers propietarios) en el filesystem. Esto no es linking, por lo que no es una violación de la GPL.



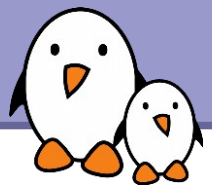
Como poblar un initramfs

Usando `CONFIG_INITRAMFS_SOURCE`
en la configuración del Kernel (`General Setup`)

- Se le puede dar un `cpio` existente
- O darle un directorio para ser archivado.

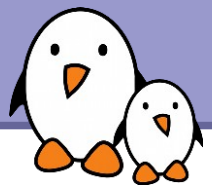
ver `Documentation/filesystems/ramfs-rootfs-initramfs.txt`
y `Documentation/early-userspace/README` en las fuentes del Kernel.

También, es http://www.linuxdevices.com/articles/AT4017834659.html hay una linda descripción de initramfs (por Rob Landley).



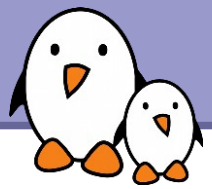
Resumen

- Para sistemas embebidos, dos soluciones interesantes:
  - Sin initramfs: Todos los drivers están incluidos en el kernel y se carga el rootfs final directamente.
  - Todo en el initramfs!



Compilando y booteando Linux

Cross-compilando el Kernel

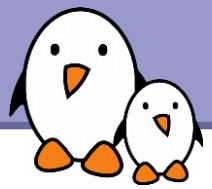


Cross-compilando el Kernel

Cuando se compila el kernel en una máquina para correr en otra

- Mucho más rápido que compilar en el target.
- Más sencillo usar las herramientas de desarrollo en el desktop.
- Para diferenciarlos de los compiladores nativos, los cross-compiladores se les cambia los nombres. Ejemplos:

```
mips-linux-gcc
m68k-linux-uclibc-gcc
arm-linux-gnueabi-gcc
```



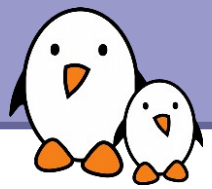
Especificar el cross-compilador

La arquitectura de la CPU y qué compilador usar se especifican con las variables `ARCH` and `CROSS_COMPILE` del `Makefile`.

- El `Makefile` define `CC = $(CROSS_COMPILE)gcc`
  Los comentarios del `Makefile` dan más detalles.
- Lo más fácil sería modificar el `Makefile`.
  Ejemplo, plataforma ARM, cross-compiler: `arm-linux-gcc`

```
ARCH          ?= arm
CROSS_COMPILE  ?= arm-linux-
```

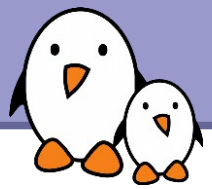
- Otras soluciones
  - Agregar `ARCH` y `CROSS_COMPILE` a la línea de comando del make
  - Definirlas como variables de entorno.
  - No olvidarse de configurar estos valores para todos los pasos. Sino el kernel se confunde!



Configurando el Kernel

```
make xconfig
```

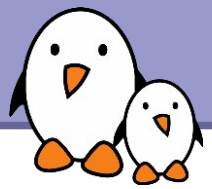
- Igual que la compilación nativa
- Las opciones van a ser diferentes
- No olvidar de poner el tipo de dispositivo correcto!



Archivos de configuración

```
assabet_defconfig       integrator_defconfig  mainstone_defconfig
badge4_defconfig        iq31244_defconfig     mx1ads_defconfig
bast_defconfig          iq80321_defconfig     neponset_defconfig
cerfcube_defconfig      iq80331_defconfig     netwinder_defconfig
clps7500_defconfig      iq80332_defconfig     omap_h2_1610_defconfig
ebsa110_defconfig       ixdp2400_defconfig    omnimeter_defconfig
edb7211_defconfig       ixdp2401_defconfig    pleb_defconfig
enp2611_defconfig       ixdp2800_defconfig    pxa255-idp_defconfig
ep80219_defconfig       ixdp2801_defconfig    rpc_defconfig
epxa10db_defconfig      ixp4xx_defconfig      s3c2410_defconfig
footbridge_defconfig    jornada720_defconfig  shannon_defconfig
fortunet_defconfig      lart_defconfig        shark_defconfig
h3600_defconfig         lpd7a400_defconfig    simpad_defconfig
h7201_defconfig         lpd7a404_defconfig    smdk2410_defconfig
h7202_defconfig         lubbock_defconfig     versatile_defconfig
hackkit_defconfig       lusl7200_defconfig
```

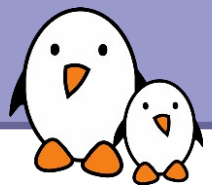
Se encuentran en `arch/arm/configs`



Usando los archivos de ejemplo

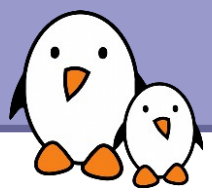
- Se proveen ejemplos de configuración para muchas archs y placas. Buscar si está la nuestra en `arch/<arch>/configs/`.
- Ejemplo: Si tenemos que usar `acme_defconfig`, corremos: `make acme_defconfig`
- Usar `arch/<arch>/configs/` es una muy buena manera de distribuir la configuración a un grupo de usuarios o desarrolladores.

Como con todos los comandos de `make`
Hay que especificar la arch
corriendo `make <machine>_defconfig`



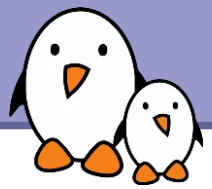
Compilando el Kernel

- Correr
  `make`
- Copiar
  `arch/<arch>/boot/zImage`
  al almacenamiento del dispositivo.
- Uno puede modificar `arch/<arch>/boot/install.sh` para que `make install` lo haga automáticamente.
- `make INSTALL_MOD_PATH=<dir>/ modules_install`
  y copiar `<dir>/lib/modules/` a `/lib/modules/` en el almacenamiento del dispositivo.

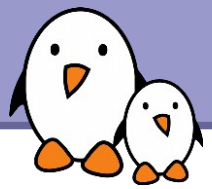


Práctica – Cross-compilando

- Configurar el entorno de desarrollo
- Configurar el `Makefile`
- Cross-compilar el kernel para una plataforma `arm`
- En esta plataforma, usar el bootloader para bootear el kernel.

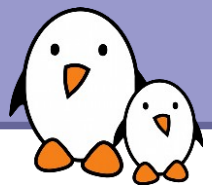


Usando módulos del kernel



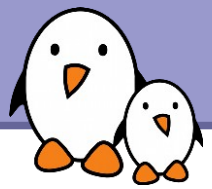
Módulos

- Módulos: Agregan una funcionalidad dada al kernel (drivers, soporte a algún filesystem, y muchas otras cosas).
- Se pueden cargar y descargar en cualquier momento, sólo cuando se necesiten.
- Sirven para desarrollar drivers sin reiniciar: cargar, probar, descargar, compilar, cargar...
- Útil para poder hacer la imagen del Kernel chica. Importante para las distribuciones de PC.
- Sirve para reducir el tiempo de booteo. No perdés tiempo iniciando drivers que no usas, o que vas a necesitar después.
- Cuidado: una vez cargado, tienen permiso completo en todo el espacio del kernel. Sin protecciones especiales.



Dependencia de los módulos

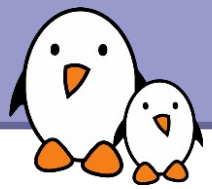
- Algunos módulos de kernel depende de otros, que necesitan ser cargados primero.
- Ejemplo: el módulo `usb-storage` depende de los módulos `scsi_mod`, `libusual` y `usbcore`.
- Las dependencias se describen en `/lib/modules/<kernel-version>/modules.dep`
  Este archivo se genera al ejecutar `make modules_install`.
- Se puede actualizar `modules.dep` a mano, corriendo (como `root`):
  `depmod -a [<version>]`



Kernel log

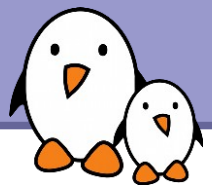
Cuando un nuevo módulo se carga, información relacionada aparece en el log del Kernel.

- El kernel guarda sus mensajes en un buffer circular. (para no ocupar toda la memoria con mensajes)
- Los mensajes están disponibles usando el comando `dmesg`. ("**d**iagnostic **mes**sa**g**e")
- Los mensajes también se muestran en la consola del sistema (se pueden filtrar los mensajes por nivel en `/proc/sys/kernel/printk`)
- Se pueden agregar cosas a su log desde userspace:
  `echo "Debug info" > /dev/kmsg`



Utilidades de módulos

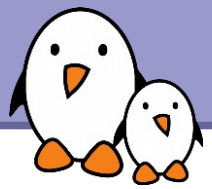
- `modinfo <module_name>`
  `modinfo <module_path>.ko`
  Pide información de un módulo: parámetros, licencia, descripción y dependencias.
  Muy útil antes de cargar un módulo.
- `sudo insmod <module_path>.ko`
  Trata de cargar un módulo. Hay que darle el path completo al archivo.



Problemas al cargar un módulo

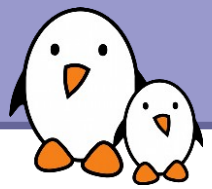
- Cuando una carga falla, `insmod` no suele dar los detalles necesarios.
- En general se encuentran en el kernel log.
- Ejemplo:

```
> sudo insmod ./intr_monitor.ko
insmod: error inserting './intr_monitor.ko': -1
Device or resource busy
> dmesg
[17549774.552000] Failed to register handler for
irq channel 2
```



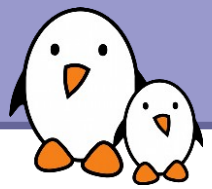
Utilidades de módulos

- `sudo modprobe <module_name>`
  Es el uso más común de `modprobe`: trata de cargar el módulo (incluyendo sus dependencias). Muchas otras opciones disponibles. Modprobe automáticamente busca en `/lib/modules/<version>/` para el módulo de ese nombre.
- `lsmod`
  Muestra una lista con los módulos cargados. Compararlo con `/proc/modules`!



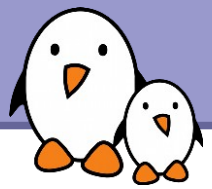
Utilidades de módulos

- `sudo rmmod <module_name>`
  Trata de descargar un módulo.
  Sólo se puede si el módulo no está en uso.
  (por ejemplo, no hay más procesos abriendo su archivo de dispositivo)
- `sudo modprobe -r <module_name>`
  Trata de descargar el módulo y todas sus dependencias que no estén siendo usadas.



Dando parámetros a los módulos

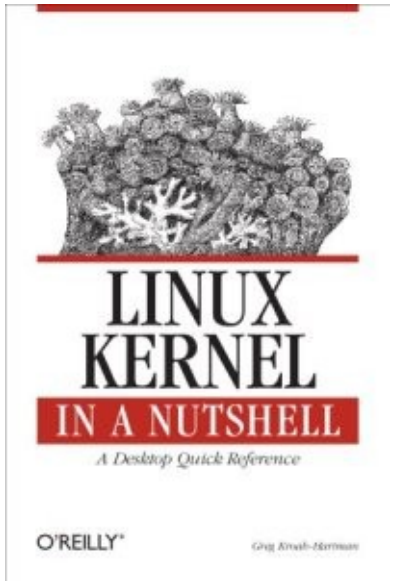
- Para conocer los parámetros disponibles:
  `modinfo snd-intel8x0m`
- Usando `insmod`:
  `sudo insmod ./snd-intel8x0m.ko index=-2`
- Usando `modprobe`:
  Configurar los parámetros en `/etc/modprobe.conf` o en otros archivos en `/etc/modprobe.d/`:
  `options snd-intel8x0m index=-2`
- Mediante la linea de comandos del kernel, cuando el módulo está compilado estático en el kernel:
  `snd-intel8x0m.index=-2`



Lecturas útiles

Linux Kernel in a Nutshell, Dec 2006

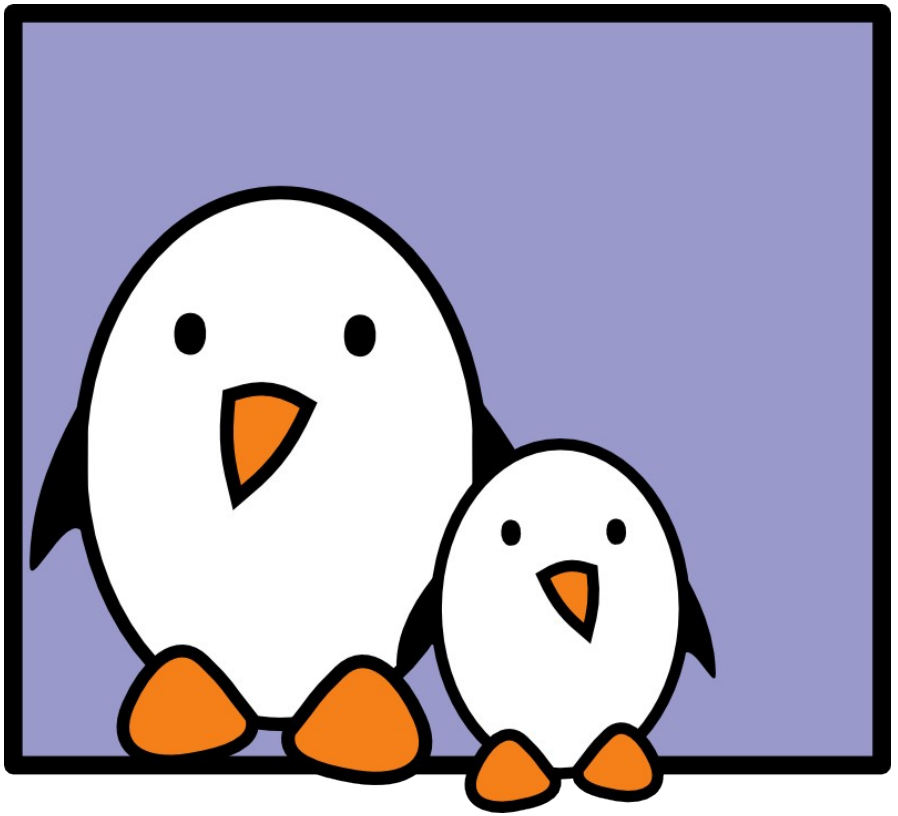
- Por Greg Kroah-Hartman, O'Reilly
  http://www.kroah.com/lkn/
- Una buena referencia en configuración, compilación y administración de las fuentes del kernel.
- Disponible gratis on-line!
  Buena compañía a los libros impresos, podemos buscar!
  Se puede bajar en pdf de:
  http://free-electrons.com/community/kernel/lkn/



SASE

Simposio Argentino de Sistemas Embebidos

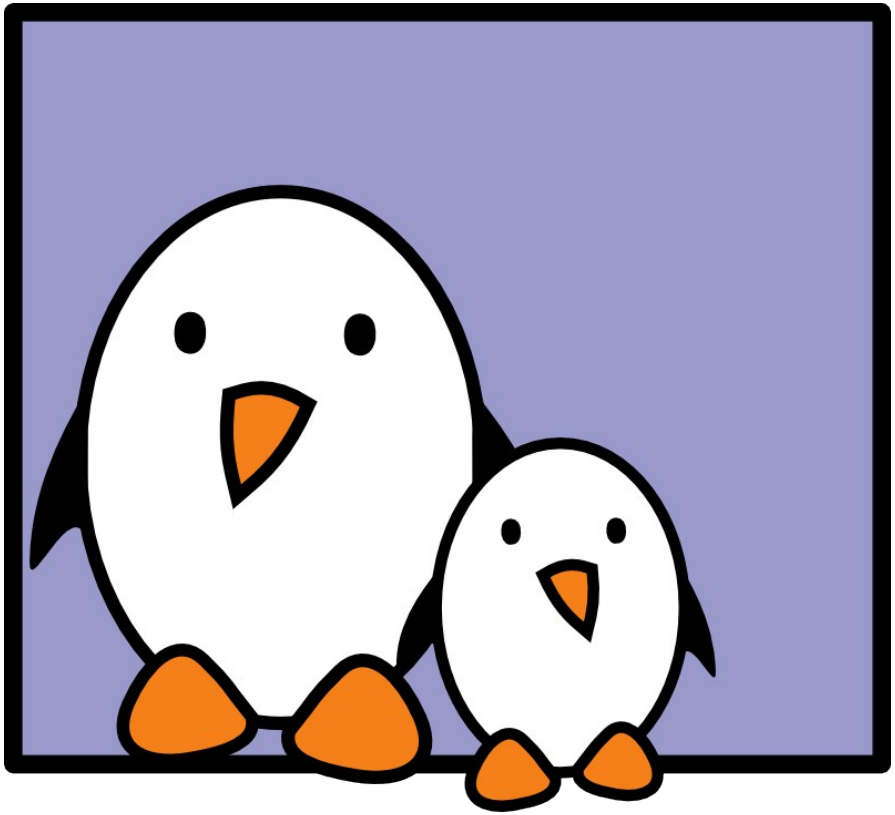
Workshop Linux Embebido

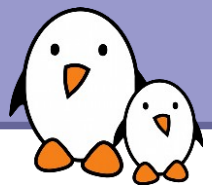


Lucas Chiesa
Joaquín de Andrés
Germán Bassi
Laboratorio
Sistemas embebidos
FIUBA

Basado en : http://free-electrons.com/docs/embedded-linux-intro

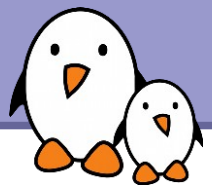
Desarrollo de sistemas Linux Embebidos



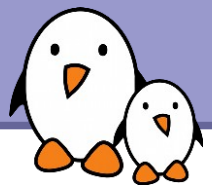


Contenidos

- Usando componentes libres
- Herramientas para usar en nuestro dispositivo
  - Redes
  - Utilidades de sistema
  - Intérpretes de lenguajes
  - Bases de datos
- Compilando el sistema
- Emuladores

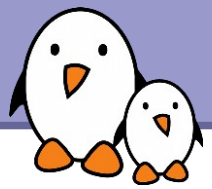


Seleccionando componentes libres para nuestro sistema embebido Linux



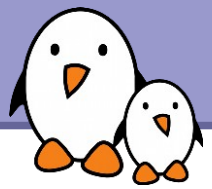
Librerías y aplicaciones de 3ros

- Una de las ventajas de usar Linux embebido es la gran cantidad de librerías y aplicaciones que uno puede utilizar para su producto.
  - Están disponibles y se pueden distribuir gratuitamente, y gracias a su naturaleza abierta, se pueden analizar y modificar para ajustar a nuestras necesidades.
- Sin embargo, reusar estos componentes en forma eficiente no es siempre una tarea fácil. Uno debe:
  - Encontrar los componentes
  - Seleccionar el más apropiado
  - Cross-compilarlo
  - Integrarlo a nuestro sistema y nuestras otras aplicaciones



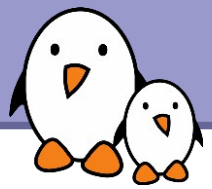
Encontrar componentes

- Freshmeat, una página de referencia para muchos proyectos libres
  http://www.freshmeat.net
- Free Software Directory
  http://directory.fsf.org
- Mirar otros sistemas Linux embebidos y ver qué usan
- Mirar los paquetes distribuidos por los sistemas de compilación de sistemas embebidos.
  - Se suelen distribuir por ser apropiados para embebidos.
- Preguntar a la comunidad o a Google
- Vamos a ver algunas de las aplicaciones comunes



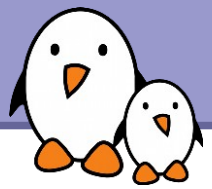
Seleccionando componentes

- No todo el software libre está en un estado listo para usarse. Hay que prestar atención a:
  - **Vitalidad** del desarrollador y la comunidad de usuarios. Esta vitalidad es la que asegura el mantenimiento a largo plazo y un buen soporte. Se puede medir mirando a la lista de correo y a la actividad en el sistema de control de versiones.
  - **Calidad** de un componente. Típicamente, si el software se usa en embebidos y tiene una comunidad de usuario importante, significa que es de una calidad razonable.
  - **Licencia**. La licencia de un componente tiene que coincidir con nuestras restricciones de licenciamiento. Por ejemplo, librerías GPL no se pueden usar en aplicaciones propietarias.
  - **Requerimientos técnicos**. Por supuesto, el componente tiene que coincidir con los requerimientos técnicos que tenemos. ¡No olvidar que le podemos agregar cosas que le falte!



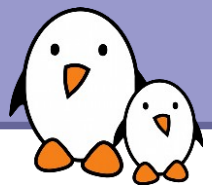
Licencias

- Todo el software libre te da las siguientes libertades:
  - Libertad de usar
  - Libertad de estudiar
  - Libertad para copiar
  - Libertad para modificar y distribuir modificaciones
- Ver http://www.gnu.org/philosophy/free-sw.html para una definición de software libre.



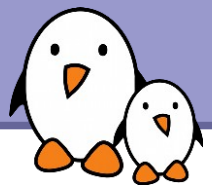
Licencias

- Se separan de dos categorías
  - Las copyleft
  - Las no copyleft
- El concepto de « copyleft » es pedir la reciprocidad en las libertades dadas a un usuario.
- El resultado es que cuando uno recibe software con copyleft y distribuye una versión modificada del mismo, tiene que hacerlo sobre la misma licencia.
  - Mismas libertades para los nuevos usuarios.
  - Trata de fomentar que se contribuyan las mejoras realizadas en lugar de mantenerlas secretas
- Las no-copyleft no tienen ese requerimientos. Podés no compartir tus cambios, pero se necesita atribución.



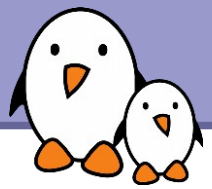
GPL

- GNU General Public License
- Cubre el ~55% de los proyectos.
  - Incluye el Kernel, Busybox y muchas otras aplicaciones
- Es una licencia copyleft
  - Requiere el trabajo derivado en la misma licencia
  - Programas que linkean con librerías GPL, tienen que ser GPL.
- Algunos programa usan la versión 2 (Linux kernel, Busybox y otros)
- Cada vez más, se adopta la versión 3, escrita en 2007
  - Impacta en sistemas embebidos. La v3 exige que el usuario pueda ejecutar la versión modificada en el dispositivo.



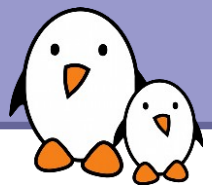
GPL: redistribución

- No hay obligaciones si no se distribuye el código
  - Podes mantener los cambios secretos hasta el día de venderlo.
- Luego se pueden distribuir copias binarias siempre y cuando:
  - Se provea el binario con el código en algún medio físico.
  - Se provea el binario con instrucciones para obtener el código, que tienen que servir por tres años.
  - Detallado en la sección 6 de la licencia GPL
- En todos los casos la atribución y la licencia se debe preservar.
  - Ver secciones 4 y 5



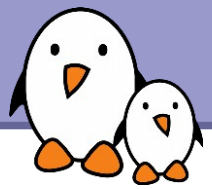
LGPL

- GNU Lesser General Public License
- Cubre el ~10% de los proyectos libres.
- Es una licencia copyleft
  - Versiones modificadas se deben distribuir con la misma licencia.
  - Pero los programas que linkean contra LGPL no necesitan mantener la licencia, pueden ser propietarios.
  - Sin embargo, el usuario tiene que tener la posibilidad de actualizar la librería. Se debe usar linkeo dinámico.
- Usado en lugar de GPL es muchas librerías, incluidas las librerías de C
  - Algunas excepciones: MySQL, o Qt <= 4.4
- También tiene dos versiones, v2 y v3



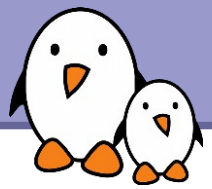
Licencias: ejemplos

- Se hacen modificaciones al Kernel (agregar drivers o adaptar a una placa), a Busybox, U-Boot u otro componente GPL
  - Debés dar las versiones modificadas bajo la misma licencia, y estar listo para distribuir el código a los distintos clientes.
- Se hacen modificaciones a la librerías de C o a otra biblioteca LPGL
  - Se debe distribuir la versión modificada bajo la misma licencia.
- Se crea una aplicación con una biblioteca LGPL
  - Podes mantener la aplicación propietaria, pero tenés que linkear dinámicamente.
- Haces modificaciones a código no copyleft
  - Podes mantener las modificaciones propietarias, pero tenés que dar crédito a los autores.



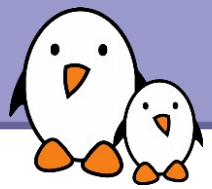
Licencias sin copyleft

- Hay muchas licencias sin copyleft, pero son todas muy similares en sus requerimientos.
- Algunos ejemplos
  - Apache (~ 4%)
  - BSD (~ 6%)
  - MIT (~ 4%)
  - Artistic license (~9 %)



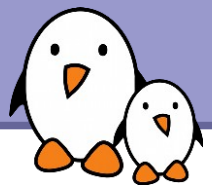
Licencia BSD

```
Copyright (c) <year>, <copyright holder>
All rights reserved.

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are met:
    * Redistributions of source code must retain the above
copyright notice, this list of conditions and the following
disclaimer.
    * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided
with the distribution.
    * Neither the name of the <organization> nor the names of
its contributors may be used to endorse or promote products
derived from this software without specific prior written
permission.

[...]
```



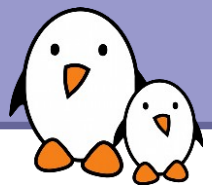
¿Es esto software libre?

- La mayoría de los proyectos usan una de ~10 licencias bien conocidas. Es bastante fácil para la mayoría de los proyectos entender su licencia.
- Si no es una conocida, leer la licencia.
- Buscar la opinión de la FSF
  http://www.fsf.org/licensing/licenses/
- O la de OSI
  http://www.opensource.org/licenses
- Debian puede ser una buena referencia también

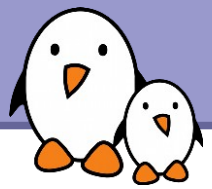


¡Respetar las licencias libres!

- El software libre no es de dominio público. Los distribuidores tienen obligaciones, según las licencias.
  - **Antes** de usar un componente, verificar que la licencia cumpla con nuestras limitaciones.
  - Estar seguro de mantener una lista de todo el software libre que se usa. La versión original y todos las modificaciones y adaptaciones bien separadas.
  - Cumplir con los requerimientos de la licencia, **antes** de distribuir nuestro producto
- Las licencias de software libre fueron aplicadas en juicios.
  - GPL-violations.org, http://www.gpl-violations.org
  - Software Freedom Law Center, http://www.softwarefreedom.org/



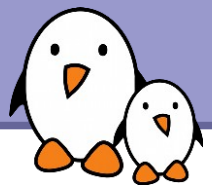
Algunas aplicaciones destacadas



Dropbear: cliente y server ssh

http://matt.ucc.asn.au/dropbear/dropbear.html

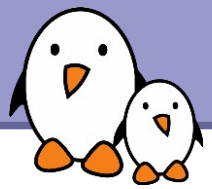
- Server ssh de muy poco footprint, para sistemas embebidos.
- Sirve tanto como cliente y como server!
- Tamaño: `110 KB`, compilado estático con uClibc en `i386`.
  (OpenSSH cliente y server: approx `1200 KB`,
  compilando dinámico con glibc en `i386`)
- Útil para:
  - Te da acceso a una terminal en el dispositivo.
  - Te permite copiar archivos al target (`scp` o `rsync -e ssh`).



Beneficios de una interfaz web

Muchos dispositivos de red, sólo tienen esa interfaz

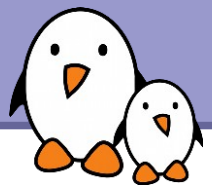
- Ejemplos: modems / routers, cámaras IP, impresoras...
- ¡No hay que hacer programas especiales para usar en las computadoras conectadas al dispositivo! ¡No hay problema soportando todos los sistemas operativos!
- Sólo se necesita escribir una página HTML estática o dinámica (posiblemente con mucho JavaScript, ejecutado en el cliente). Es una manera simple de poder cambiar la configuración del dispositivo.
- Reduce el costo de hardware. No necesitamos agregar LCDs, menos botones. Sólo necesitamos un poco más de almacenamiento.



thttpd

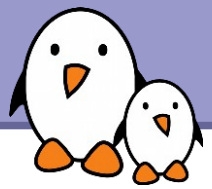
Tiny/Turbo/Throttling HTTP server
http://acme.com/software/thttpd/

- Simple
  Implementa HTTP/1.1 mínimo (o quizá un poquito más)
  Simple para configurar y usar.
- Pequeño
  Tamaño pequeño: 88K,
  Apache: 264K
  Muy poco uso de memoria, no hace forks y tiene cuidado con el uso de memoria.
- Portable
  Compila bien en la mayoría de los sistemas Unix.
- Rápido
  Casi tan rápido como los servers más importantes.
- Seguro
  Diseñado para proteger al sistema que lo ejecuta de ataques.
- Otros servers: BusyBox, Lighttpd, BOA



Intérpretes

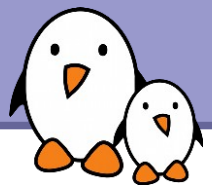
- Hay interpretes para los lenguajes de scripts más comunes, útiles para:
  - Desarrollo de aplicaciones
  - Desarrollo de servicios web.
  - Scripting
- Lenguajes soportados
  - Lua
  - Python
  - Perl
  - Ruby
  - TCL
  - PHP



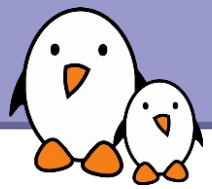
Bases livianas - SQLite

http://www.sqlite.org

- SQLite es una pequeña librería de C que implementa un motor de SQL autocontenido, embebible, liviano, y sin configuración.
- Es el motor de bases de datos preferido para los sistemas embebidos.
  - Se puede utilizar como una librería normal.
  - Se puede embeber directamente en una aplicación, incluso en una propietaria ya que SQLite es liberada al dominio público.

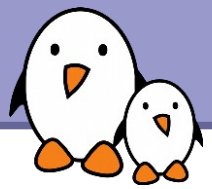


Ejemplos de dispositivos reales



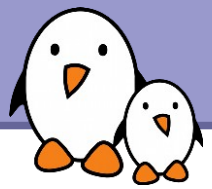
Aplicaciones industriales

- En muchas aplicaciones industriales, el sistema es responsable de observar y controlar un dispositivo.
- Dicho sistema es usualmente relativamente simple en términos de componentes:
  - Kernel
  - BusyBox
  - Librería de C
  - Aplicaciones que usan directamente la librería de C, posiblemente utilizando las capacidades real-time del Kernel Linux.
  - Posiblemente un servidor Web para el control remoto del equipo. También puede ser otro servidor implementando un protocolo propio.



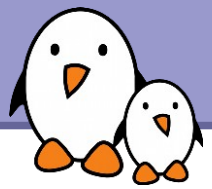
Portarretratos digital: requerimientos

- Ejemplo tomado de una conferencia por Matt Porter, Embedded Alley en ELC 2008
- Hardware: ARM SoC con DSP, audio, 800x600 LCD, MMC/SD, NAND, botones, parlantes
- El portarretratos tiene que poder:
  - Dibujar en el LCD
  - Detectar la conexión de una SD, notificar a las aplicaciones de esto para que pueden crear un catálogo de imágenes en la memoria.
  - Interfaz 3D moderna con lindas transiciones.
  - Navegación por medio de botones.
  - Reproducción de audio (MP3, playlists, ID3 tag)
  - Escalar y rotar JPEG



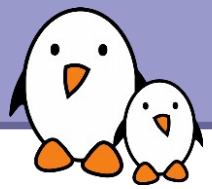
Portarretratos digital: componentes

- Sistema base
  - Componentes presentes en prácticamente todos los sistemas Linux embebidos.
  - El bootloader U-Boot
  - El Kernel Linux
    - Drivers para SD/MMC, framebuffer, sonido, dispositivos de entrada
  - Busybox
  - Sistema para compilar el sistema, utilizaron OpenEmbedded
  - Componentes: **U-Boot, Kernel, BusyBox**



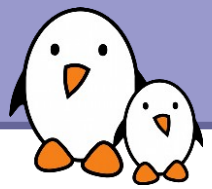
Portarretratos digital: componentes

- Manejo de eventos e inserción de una memoria SD.
  - udev recibe el evento del Kernel, crea los nodos de los dispositivos y envía los eventos a HAL.
  - HAL mantiene una base de datos de dispositivos disponibles y provee una interfaz D-Bus.
  - D-Bus para conectar HAL con la aplicación. La aplicación se suscribe a los eventos de HAL por medio de D-Bus y de esta forma se entera cuando hay nuevos eventos de hardware.
  - Componentes: **udev, hal, dbus**



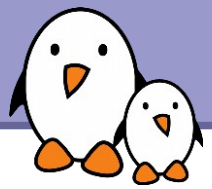
Portarretratos digital: componentes

- Abrir imágenes JPEG
  - **libjpeg** para decodificar las imágenes.
  - **jpegtran** para escalar y rotar las imágenes.
  - FIM (Fbi Improved) para dithering
- Soporte MP3
  - **libmad** para reproducir
  - **libid3** para leer los tags mp3
  - **libm3u** para soportar playlists
  - Se usan componentes provistos por el fabricante para utilizar el DSP para reproducir MP3.

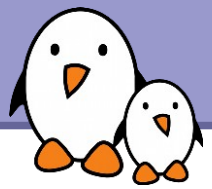


Portarretratos digital: componentes

- Interfaz 3D
  - Vincent, una implementación libre de OpenGL ES
  - Clutter, API de alto nivel para desarrollar aplicaciones 3D.
- La aplicación en si:
  - Maneja eventos de la SD
  - Usa las librerías de JPEG para decodificar y dibujar las imágenes.
  - Recibe los eventos de entrada de los botones y dibujar una interfaz OpenGL programada utilizando Clutter.
  - Maneja las configuraciones de usuario.
  - Reproduce los archivos MP3 con las librerías relacionadas.
  - Pasa las fotos en forma de diapositivas.

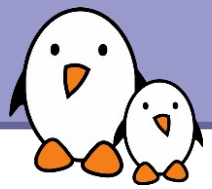


Compilando el sistema



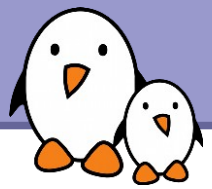
Compilando el sistema

- Objetivos
  - Integrar todos los componentes de software, tanto los de terceros como los propios, obteniendo un rootfs funcional.
  - Involucra descargar, extraer, configurar, compilar e instalar todos los componentes. Se puede necesitar resolver problemas y adaptar las configuraciones.
- Muchas soluciones
  - Manualmente
  - Herramientas de compilación de sistema.
  - Distribuciones o filesystems pre armados.



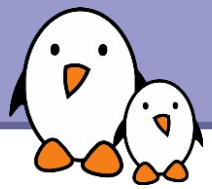
Compilando el sistema: manualmente

- Implica realizar todos los pasos mencionados, a mano.
- Todas las bibliotecas y dependencias se deben configurar, compilar e instalar en el orden correcto.
- Algunas veces, el sistema de compilación usado por librerías o aplicaciones no facilita la cross-compilación, por lo que puede ser necesario adaptarlo.
- No hay infraestructura para reproducir una compilación desde cero. Puede ser problemático si necesitamos cambiar un componente o si otra persona retoma el proyecto, etc..



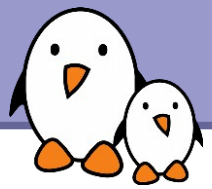
Compilando el sistema: manualmente

- No es recomendado para proyectos que van a entrar en producción.
- Sin embargo, usando las herramientas automáticas, de vez en cuando nos vamos a encontrar con problemas particulares.
- Tener un entendimiento básico de cómo se compila un sistema manualmente resulta de mucha utilidad para arreglar los problemas encontrados con las herramientas automáticas.
  - Vamos a analizar una herramienta automática y usarla en una práctica.



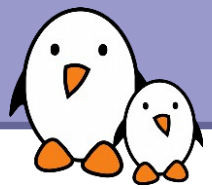
Cimientos del sistema

- Un rootfs básico necesita por lo menos:
  - Una jerarquía tradicional de directorios, con `/bin`, `/etc`, `/lib`, `/root`, `/usr/bin`, `/usr/lib`, `/usr/share`, `/usr/sbin`, `/var`, `/sbin`
  - Un conjunto de utilidades básicas, que proveen por lo menos el programa init, un shell y otros comandos tradicionales de Unix. Usualmente es provisto por BusyBox.
  - La librería de C y otras librerías relacionadas (thread, math, etc.), instaladas en `/lib`
  - Unos pocos archivos de configuración, como `/etc/inittab`, y scripts de inicialización en `/etc/init.d`
- Sobre estos cimientos comunes a casi todos los sistemas, podemos agregar componentes nuestros o de terceros.



Herramientas: principios

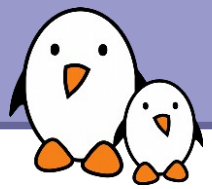
- Existen diferentes herramientas para automatizar el proceso de compilar el sistema target, incluyendo el sistema, Kernel y algunas veces el toolchain.
- Ellas, automáticamente descargan, configuran, compilan e instalan todos los componentes en el orden correcto. Muchas veces, aplican parches para arreglar problemas por cross-compilar.
- Ya contienen una gran cantidad de paquetes que deberían cubrir todos los requerimientos principales, y son fácilmente extensibles.
- La compilación se transforma en algo reproducible, lo que permite cambiar componentes, actualizarlos, arreglar bugs, más fácilmente.



Herramientas disponibles

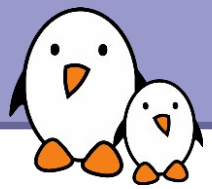
Muchas opciones

- **Buildroot**, desarrollado por la comunidad
  http://www.buildroot.net
- **PTXdist**, desarrollado por Pengutronix
  http://www.pengutronix.de/software/ptxdist/index_en.html
- **OpenWRT**, originalmente un fork de Buildroot para routers WiFi, ahora un proyecto más genérico.
  http://www.openwrt.org
- **LTIB**, desarrollado principalmente por Fresscale. Buen soporte para placas Freescale (algo de NXP). Comunidad pequeña.
  http://www.bitshrine.org/
- **OpenEmbedded**, más flexible pero más complejo
  http://www.openembedded.org
- Herramientas específicas de un fabricante (de silicio o soporte en Linux)



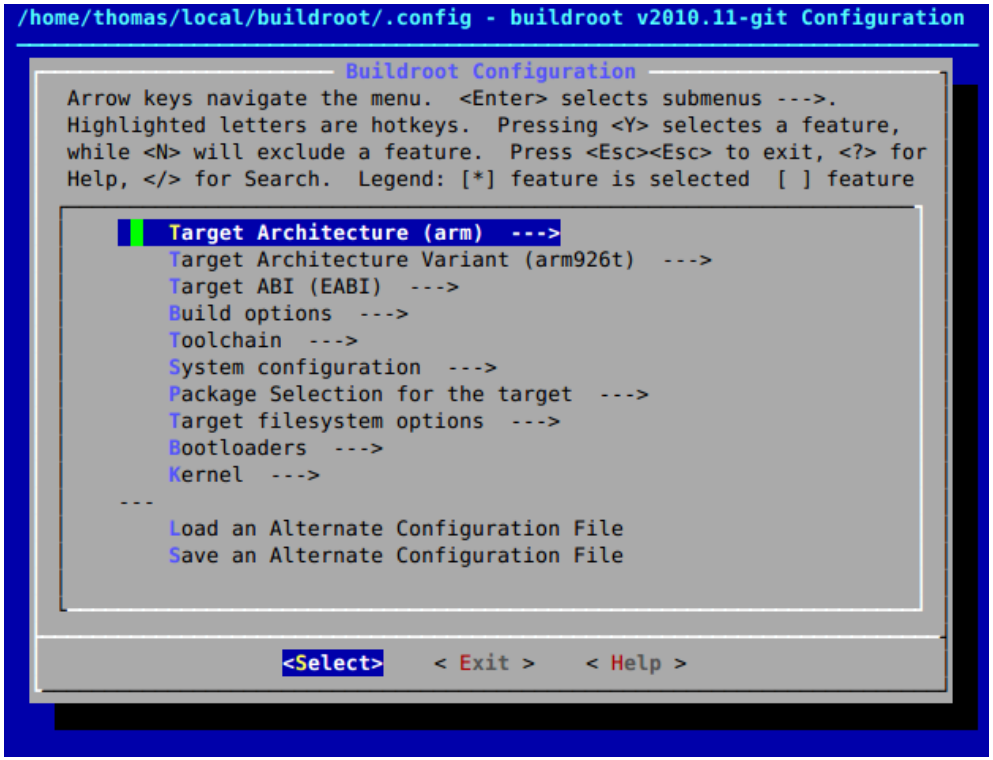
Buildroot

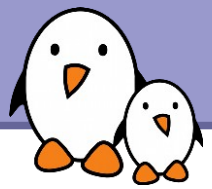
- Permite compilar el toolchain, un root fs con muchas aplicaciones y librerías, un bootloader y una imagen del Kernel.
- Soporta compilar toolchains sólo con uClibc, pero puede usar uno externo con glibc.
- Más de 500 aplicaciones y librerías integradas. Desde las cosas más básicas hasta stacks complejos como: X.org, Gstreamer, Qt, Gtk, Webkit, etc.
- Buen soporte para sistemas pequeños y medianos, con funciones fijas.
  - No soporta generación de paquetes (.deb or .ipk)
  - Necesita una recompilación completa por casi cualquier cambio de configuración.
- Comunidad activa, un release cada 3 meses.



Buildroot

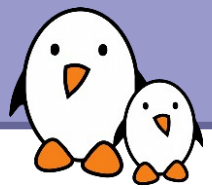
- La configuración se realiza mediante una interfaz *`*config`* similar a la del Kernel
  `make menuconfig`
- Permite definir
  - Arquitectura y CPU
  - Configuración del toolchain
  - Conjunto de aplicaciones y librerías para agregar
  - Imágenes de FS para generar
  - Conf. del Kernel y del Bootloader
- Compila todo ejecutando:
  `make`





OpenEmbedded

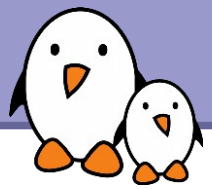
- El sistema más versátil y poderoso para armar un sistema Linux Embebido.
  - Una colección de recetas “recipes” (archivos `.bb`)
  - Una herramienta que procesa esas recetas: `bitbake`
- Integra más de 2000 aplicaciones y librerías, es altamente configurable, puede generar paquetes (`.ipk`), no recompila todo al cambiar una configuración.
- La configuración se hace editando unos cuantos archivos de configuración.
- Bueno para sistemas más grandes o para gente que busca configurabilidad y extensibilidad.
- Desventajas: no hay releases estables, difícil de aprender, muy lento en la primera compilación.



Distribuciones

Debian GNU/Linux, http://www.debian.org

- Disponible para ARM, MIPS y PowerPC
- Provee un sistema listo para usar con todo el software que puedas necesitar.
- Mucha flexibilidad gracias al sistema de paquetes. Sólo sirve si hay mucho almacenamiento (> 300 MB) y RAM (> 64 MB).
- El software se compila en forma nativa por omisión.
- Podes armar tus propias imágenes en x86 usando el comando `debootstrap`.
- Emdebian es un proyecto para hacer Debian mejor para embebidos. Utiliza los paquetes de Debian, reduciendo las dependencias, con menos configuración, saca la documentación, soporta uClibc...Ver http://emdebian.org.



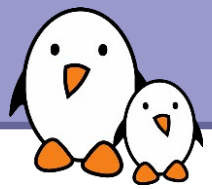
Distribuciones

Ubuntu GNU/Linux

- Basado en Debian, con algunas “mejoras”
- Un nuevo release cada 6 meses,
  soporte por 18 meces o hasta 3 años.
- Suportado en ARM, >= Cortex A8. Provee binarios Thumb2.
  Pero no soporta Neon todavía.
- Buena solución para dispositivos multimedia.

Otros

- Fedora también soporta ARM, pero no es muy mantenido.



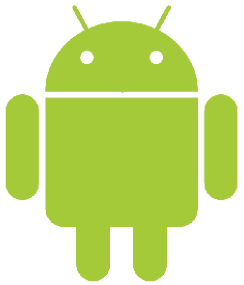
Distribuciones embebidas

Existen distribuciones específicas para embebidos

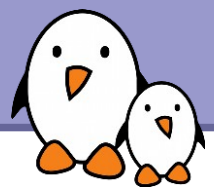
- Meego: http://meego.com/
  Apunta a celulares, reproductores de multimedia, netbooks, Tvs, IVI.
  Soportado por Intel y Nokia (más o menos).

MeeGo™

- Android: http://www.android.com/
  Distribución de Google para teléfonos y tablet PCs.
  Salvo el Kernel, un userspace muy diferente de otras distros. Muy exitosa, muchas aplicaciones disponibles (muchas propietarias).

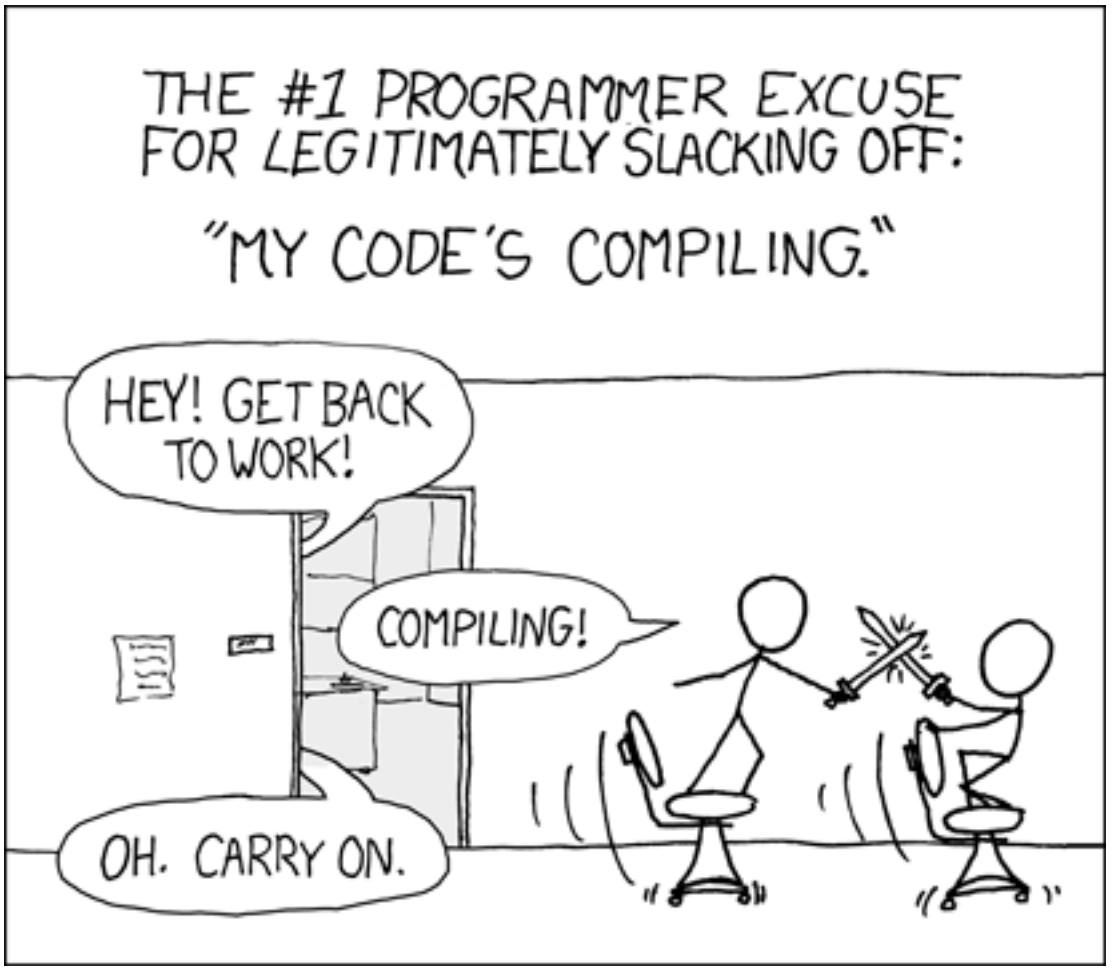


- Ångström: http://www.angstrom-distribution.org/
  Apunta a PDAs y MIDs (Siemens Simpad...)
  Binarios para `arm` little endian.

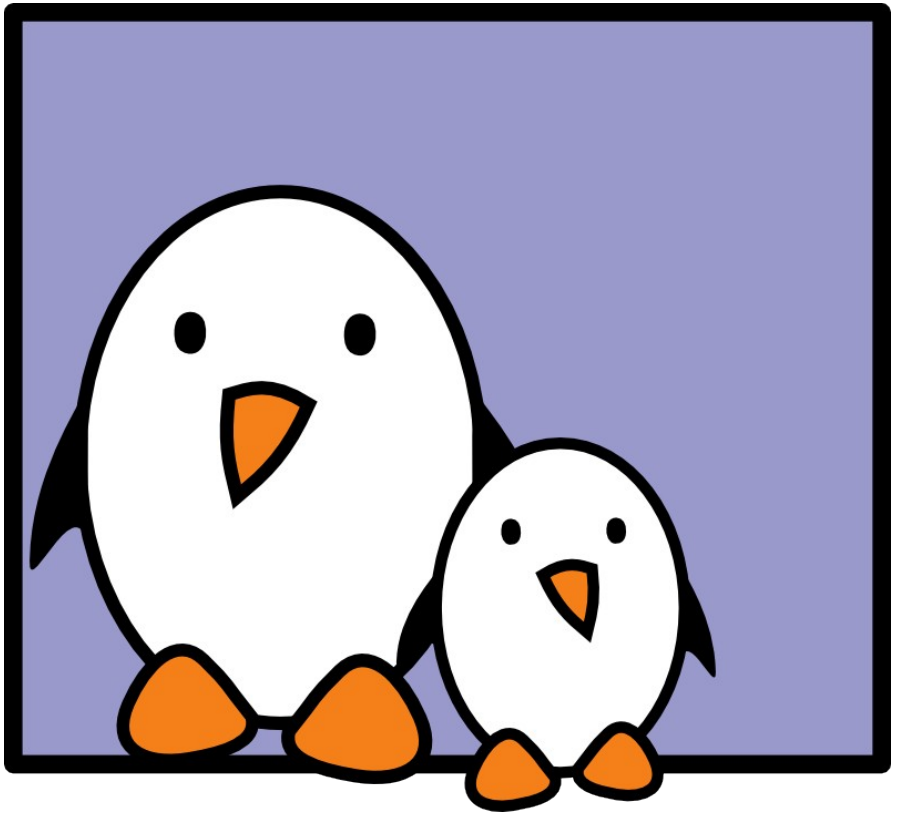


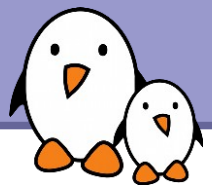
Práctica – Buildroot

- Compilar un sistema usando Buildroot.
- Observar que tan fácil es!



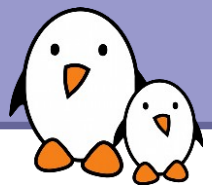
Sistemas de archivos de bloques





Block vs. Flash

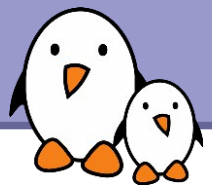
- Los dispositivos de almacenamiento se clasifican en dos categorías: **block devices** y **flashes**
  - Son manejados por subsistemas diferentes y usan sistemas de archivos diferentes.
- **Block devices** se pueden leer y escribir de a bloques, sin borrar y no se gastan por usarlos muchas veces.
  - Discos rígidos, floppy, RAM.
  - Pendrives USB, Compact Flash, SD, son basados en flash, pero tienen integrados un controlador que emula un dispositivo de bloque.
- **Flash**, necesitan ser borradas, normalmente en bloques más grandes que el tamaño de “bloque”. Llamado *eraseblock*
  - NOR y NAND flash



Block filesystems tradicionales

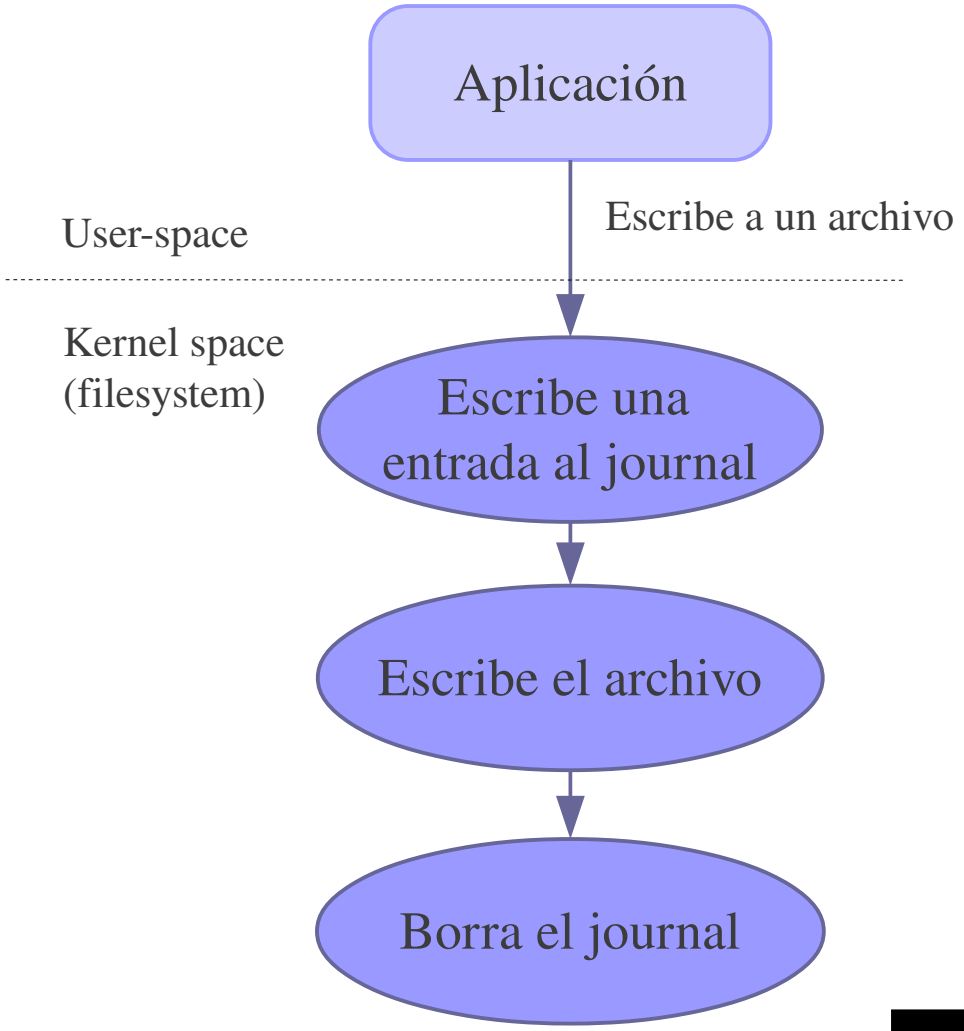
Filesystems tradicionales

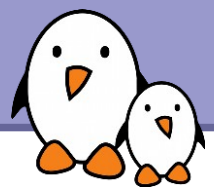
- Pueden quedar en un estado no coherente después de un problema del sistema o corte de luz. Requiere un chequeo completo al reiniciar
- `ext2`: filesystem tradicional de Linux
  (se repara con `fsck.ext2`)
- `vfat`: Windows filesystem tradicional
  (se repara con `fsck.vfat` en GNU/Linux o `Scandisk` en Windows)



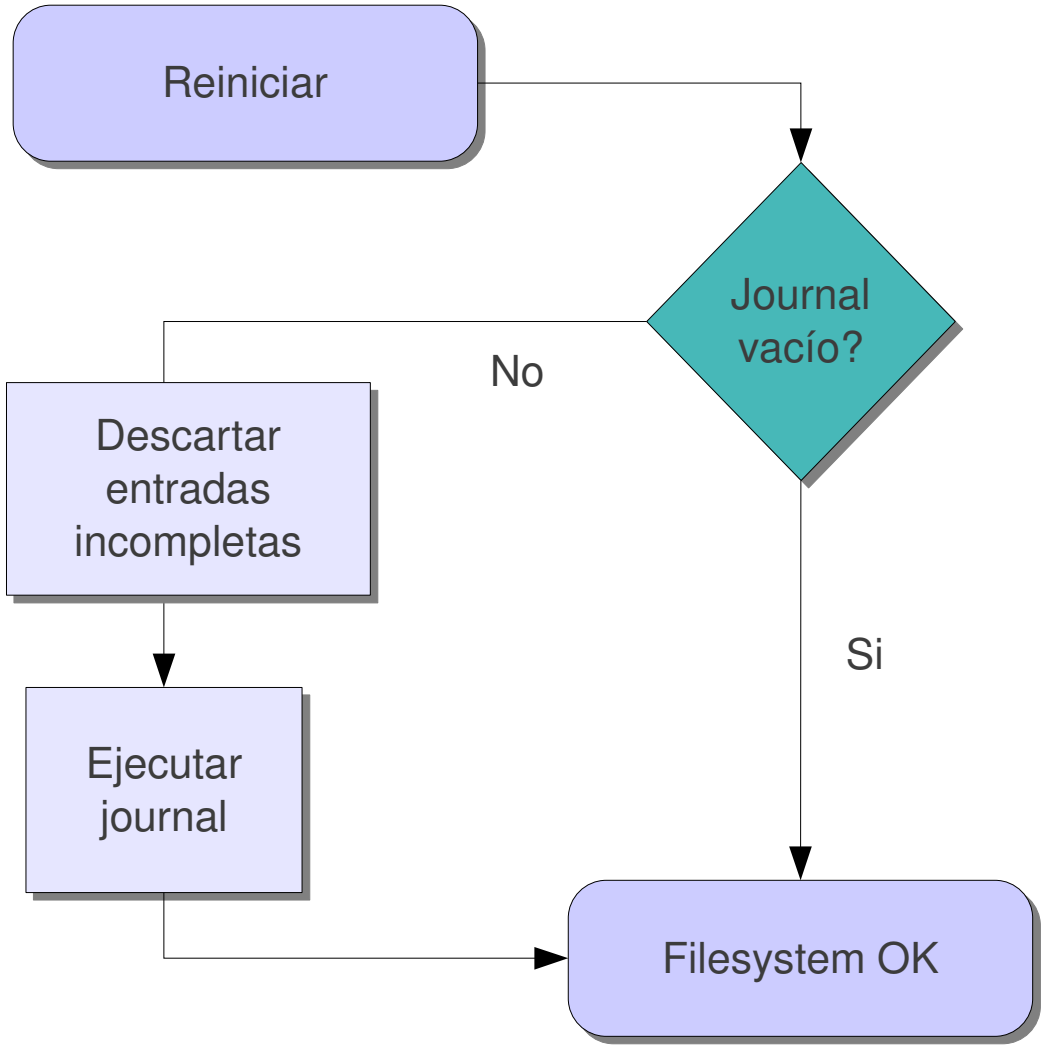
Filesystems con journals

- Diseñados para permanecer en estados correctos, incluso después de cortes de luz.
- Todas las escrituras son primero escritas al journal antes de ser pasadas a los archivos.

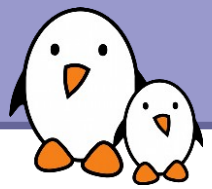




Recuperación después de un corte



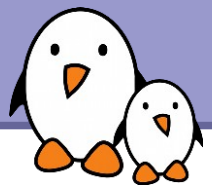
- Gracias al journal el sistema de archivos nunca se deja en un estado corrupto.
- Igualmente se pueden perder los últimos cambios antes del apagón.



filesystems de bloques con journals

filesystems de bloques con journals:

- `ext3`: `ext2` con una extensión para usar journal
  `ext4`: la nueva generación, con muchas mejoras. Ya se puede usar y es el fs por omisión en distros modernas.
- El Kernel Linux soporta muchos otros sistemas de archivos: `reiserFS`, `JFS`, `XFS`, etc. Cada uno con sus características propias, pero en general orientados a aplicaciones particulares.
- `btrfs` (“Butter FS”)
  La próxima generación de los ext. En el Kernel, pero todavía experimental.

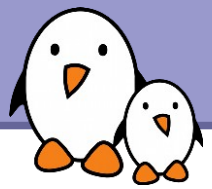


Squashfs

Squashfs: http://squashfs.sourceforge.net

- Reemplaza a un fs anterior: Cramfs! Es para hacer fs read-only.
- Tamaño máximo del fs: 2^{64} bytes.
- Logra alta compresión y alto desempeño.
  Soporta tamaños de bloque de hasta 64 K (en lugar de 4K), tiene mayor compresión y hasta detecta archivos duplicados!
- Incluido en el Kernel desde 2.6.29. Se usan parches para las anteriores.

Benchmarks: (3 veces más chicos que ext3, y 2-4 veces más rápido)
http://elinux.org/Squash_Fs_Comparisons



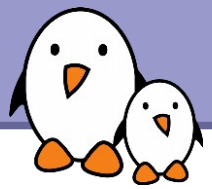
tmpfs

Útil para mantener datos en RAM: archivos de log, información temporal...

- No usar ramdisks! Tienen muchos problemas: tamaño fijo, el espacio libre no se puede usar como RAM, archivos duplicados en RAM (en el dispositivo y en el cache de archivos)!
- Configuración de tmpfs: `File systems` -> `Pseudo filesystems`
  Vive en el cache de archivos. No pierde RAM: se agranda y se achica para acomodar a los archivos, no duplica y puede mandar páginas a swap si es necesario.
- Cómo usarlo: seleccionar diferentes nombres para los “dispositivos” para reconocerlo. Ejemplos:

```
mount -t tmpfs varrun /var/run
mount -t tmpfs udev /dev
```

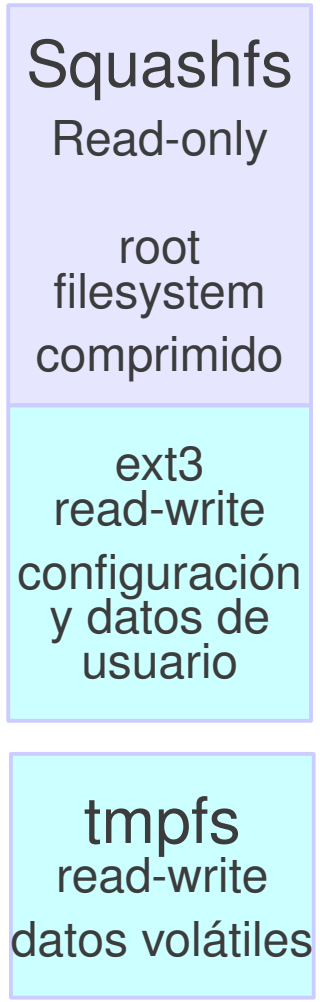
Ver `Documentation/filesystems/tmpfs.txt`.



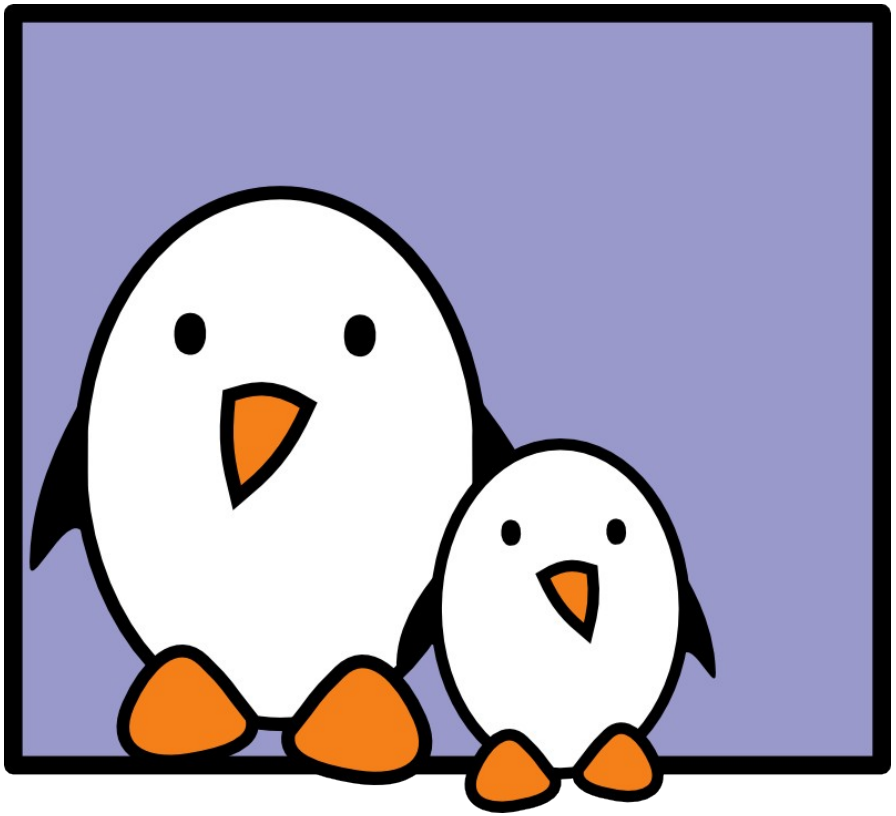
Usando read only y read write

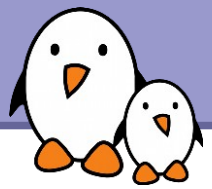
Es una buena idea separar el uso de los fs de bloques

- Una partición comprimida read only (Squashfs)
  Típicamente para el root fs (binarios, kernel...).
  Compresión ahorra espacio. Read-only previene errores y corrupción de datos.
- Una partición read-write con un fs con journal (como ext3)
  Usado para guardar datos de configuración.
  Garantiza la integridad del fs después de apagones.
- Guardar en RAM archivos temporales, como logs (tmpfs)



Filesystems Flash





El subsistema MTD

MTD: Memory Technology Devices (flash, ROM, RAM)

Interfaz de filesystems de Linux

Módulos de “usuario”

UBI | jffs2 | Char device | Block device

yaffs2 | Read-only block device

Flash Translation Layers para emular dispositivos de bloques

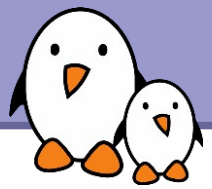
algoritmos patentados!

FTL | NFTL | INFTL

Drivers de chips MTD

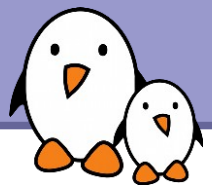
NAND flash | DiskOnChip flash | ROM chips | NOR flash | RAM chips

Hardware



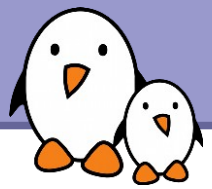
Dispositivos MTD

- Son visibles en `/proc/mtd`
- El driver **mtdchar** crea un dispositivo de caracteres para cada dispositivo MTD en el sistema.
  - Usualmente llamado `/dev/mtdX`, major 90. minors pares para acceso rw y minors impares para acceso ro.
  - Provee `ioctl()` para borrar y manejar la flash
  - Usado por *mtd-utils*
- El driver **mtdblock** crea un dispositivo de bloque por cada MTD del sistema.
  - Usualmente llamado `/dev/mtdblockX`, major 31. El minor es el número del MTD correspondiente.
  - Permite leer y escribir en bloques. No soporta sectores malos ni *wear leveling.*



Haciendo particiones en MTD

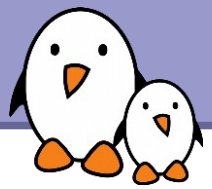
- Usualmente se hacen particiones en los MTD
  - Permite usar diferentes áreas de la flash con diferentes propósitos: FSRO, FSRW, backup, bootloader, kernel, etc.
- No tienen su propia tabla de particiones. Las particiones se deben describir externamente.
  - Hard-coded en el código del Kernel
  - Especificado en la línea de comandos del Kernel
- Cada partición crea un dispositivo MTD diferente.
  - Diferencias con el esquema de nombres de los dispositivos de bloques. (hda3, sda2)
  - `/dev/mtd1` puede ser la segunda partición del primer MTD o la primera del segundo.



Haciendo particiones en MTD

Las particiones MTD son definidas en el Kernel, en la definición de las placas: `arch/arm/mach-at91/board-usb-a9263.c`
Ejemplo:

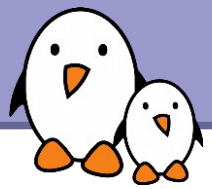
```
static struct mtd_partition __initdata ek_nand_partition[] = {
        {
                .name   = "Linux Kernel",
                .offset = 0,
                .size   = SZ_16M,
        },
        {
                .name   = "Root FS",
                .offset = MTDPART_OFS_NXTBLK,
                .size   = 120 * SZ_1M,
        },
        {
                .name   = "FS",
                .offset = MTDPART_OFS_NXTBLK,
                .size   = 120 * SZ_1M,
        }
};
```



Modificando particiones MTD

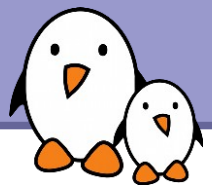
- Por suerte las particiones MTD se pueden definir por la línea de comandos.
- Primero hay que encontrar el nombre de los dispositivos MTD. Mirando el log del Kernel:

```
NAND device: Manufacturer ID: 0xec, Chip ID:
0xda (Samsung NAND 256MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 2000 at 0x0fa00000
Creating 3 MTD partitions on "atmel_nand":
0x00000000-0x01000000 : "Linux Kernel"
0x01000000-0x08800000 : "Root FS"
0x08800000-0x10000000 : "FS"
```



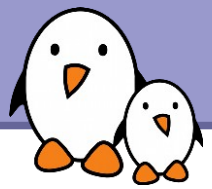
Modificando particiones MTD

- Se puede usar el parámetro de Kernel `mtdparts`
- Ejemplo:
  `mtdparts=atmel_nand:2m(kernel)ro,1m(rootfs)ro,-(data)`
- Recién definimos 3 particiones en el dispositivo `atmel_nand`:
  - `kernel` (2M)
  - `rootfs` (1M)
  - `data`
- Los tamaños de las particiones tienen que ser múltiplos del tamaño de los *eraseblock*.
  Se pueden usar tamaños en base 16. Recuerden:
  `0x20000` = 128k, `0x100000` = 1m, `0x1000000` = 16m
- `ro` crea la partición como de sólo lectura
- `-` significa todo el espacio restante.



mtd-utils

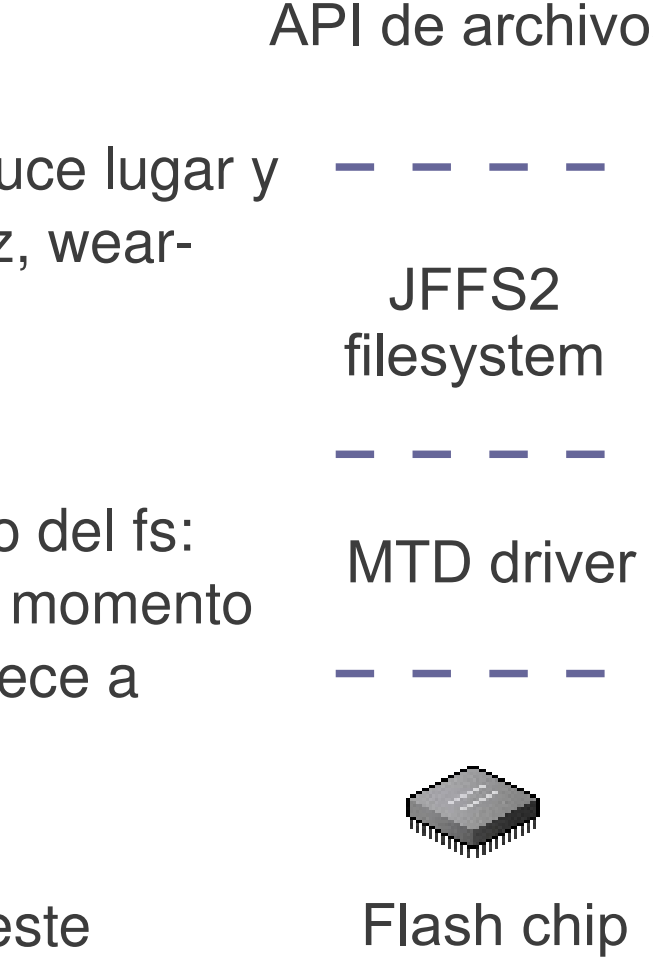
- Un conjunto de utilidades para manejar MTDs
  - `mtdinfo` da información detallada de un MTD
  - `flash_eraseall` borra completamente un MTD
  - `flashcp` para escribir a NOR flash
  - `nandwrite` para escribir a NAND flash
  - Utilidades UBI
  - Herramientas para crear imágenes de fs de flash: `mkfs.jffs2`, `mkfs.ubifs`
- Usualmente se encuentran en el paquete mtd-utils de la distribución.
- Ver http://www.linux-mtd.infradead.org/

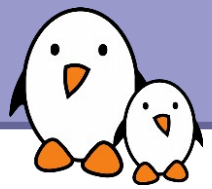


jffs2

http://www.linux-mtd.infradead.org/doc/jffs2.html

- El estándar actual para MTDs
- Lindas funciones: compresión on the fly (reduce lugar y operaciones de I/O), tolerante a cortes de luz, wear-leveling y ECC.
- Desventajas: no escala bien
  - El tiempo de montaje depende de tamaño del fs: el Kernel tiene que escanear todo el fs al momento de montar, para saber qué bloque pertenece a cada archivo.
  - Se necesita usar la opción del Kernel `CONFIG_JFFS2_SUMMARY` para reducir este problema. Guardar esta info en flash. (de 16s a 0.8s para una partición de 128 MB).

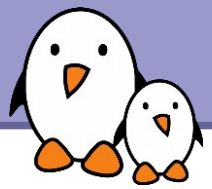




jffs2 – Cómo usarlo

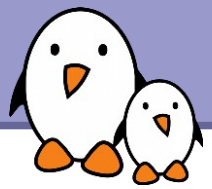
En el **target**

- Se necesita `mtd-utils` del proyecto MTD, o su variante para Busybox
- Borrar y formatear una partición jffs2:
  `flash_eraseall -j /dev/mtd2`
  Montar la partición:
  `mount -t jffs2 /dev/mtdblock2 /mnt/flash`
  Agregar el contenido escribiendo los archivos.
  (copiarlo por red o de otro almacenamiento)
- Otra posibilidad: usar una imagen jffs2:
  `flash_eraseall /dev/mtd2`
  `nandwrite -p /dev/mtd2 rootfs.jffs2`



Cómo crear la imagen jffs2

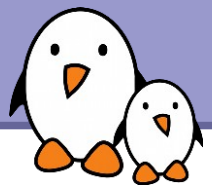
- `mkfs.jffs2:` comando disponible en el paquete `mtd-utils`. Cuidado: a diferencia de algunos comandos `mkfs`, no crea el filesystem sino una imagen.
- Primero, encontrar el tamaño *eraseblock*, U-Boot `nand info`:
  `Device 0: NAND 256MiB 3,3V 8-bit, sector size 128 KiB`
- Luego crear la imagen en el desktop:
  ```
  mkfs.jffs2 --pad --no-cleanmarkers
  --eraseblock=128 -d rootfs/ -o rootfs.jffs2
  ```
- La opción `--pad` completa la imagen hasta el final del último *ereaseblock.*
- No hay problema si la partición jffs2 es más chica que la partición. El jffs2 va a usar toda la partición de todas formas.
- La opción `--no-cleanmarkers` es sólo para NAND flash.



Montando jffs2 en el host

Útil para editar imágenes `jffs2` en el equipo de desarrollo. Es una tarea un poco compleja, esto es un ejemplo:

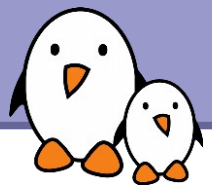
- Primero necesitamos el tamaño del *ereaseblock* que usamos para crear la imagen jffs2. Asumimos que es 128KiB (131072 bytes).
- Creamos un dispositivo de bloque con la imagen
  ```
  losetup /dev/loop0 root.jffs2
  ```
- Emulamos un MTD desde un block usando el módulo de Kernel block2mtd
  ```
  modprobe block2mtd block2mtd=/dev/loop0,131072
  ```
- Finalmente, montamos el fs (crear `/mnt/jffs2` si es necesario)
  ```
  mount -t jffs2 /dev/mtdblock0 /mnt/jffs2
  ```



Inicializando jffs2 desde U-Boot

Podemos no querer tener `mtd-utils` en el target!

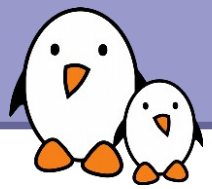
- Creamos la imagen jffs2 en el desktop
- En la línea de comandos de U-Boot:
  - Bajamos la imagen jffs2 a RAM usando `tftp`.
    O la copiamos desde almacenamiento externo (FAT en USB, por ejemplo)
  - La escribimos dentro de la partición MTD
    (las instrucciones exactas depende del tipo de memoria, NOR o NAND, las mismas que se usaron para escribir el Kernel). Estar seguros de escribir sólo el tamaño de la imagen y no más!
  - Si se usa jffs2 como root agregar `root=/dev/mtdblock<x>` y `rootfstype=jffs2` a los argumentos del kernel.
- Limitación: necesitás separar la imagen jffs2 en varios archivos si es más grande que el bloque de RAM.



UBI

Unsorted Block Images

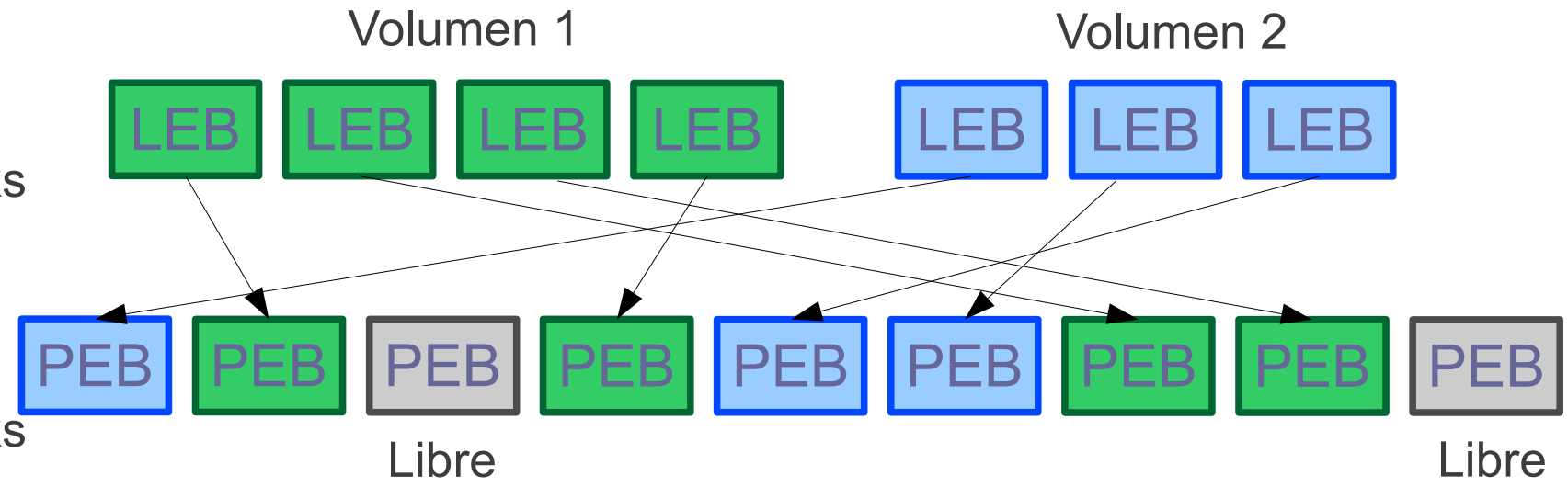
- http://www.linux-mtd.infradead.org/doc/ubi.html
- Un sistema de administración de volúmenes sobre MTD.
- Permite crear diferentes volúmenes lógicos y repartirlos entre todos los dispositivos físicos.
- Se encarga de administrar los *eraseblocks* y wear leveling. Facilita implementar los filesystems.

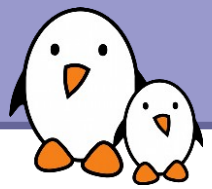


UBI

UBI
Logical
Erase Blocks

MTD
Physical
Erase Blocks

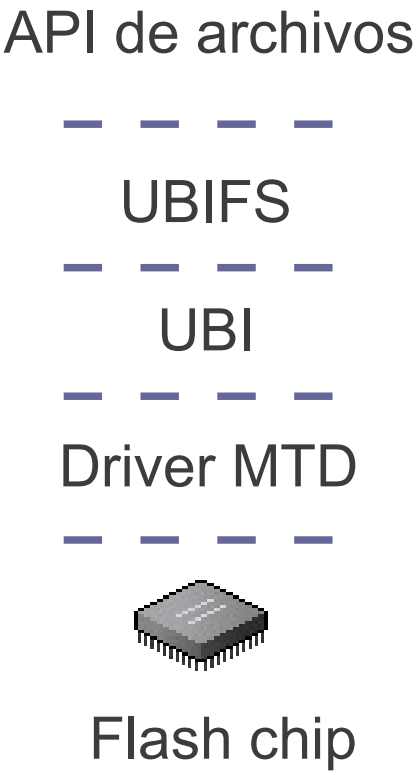




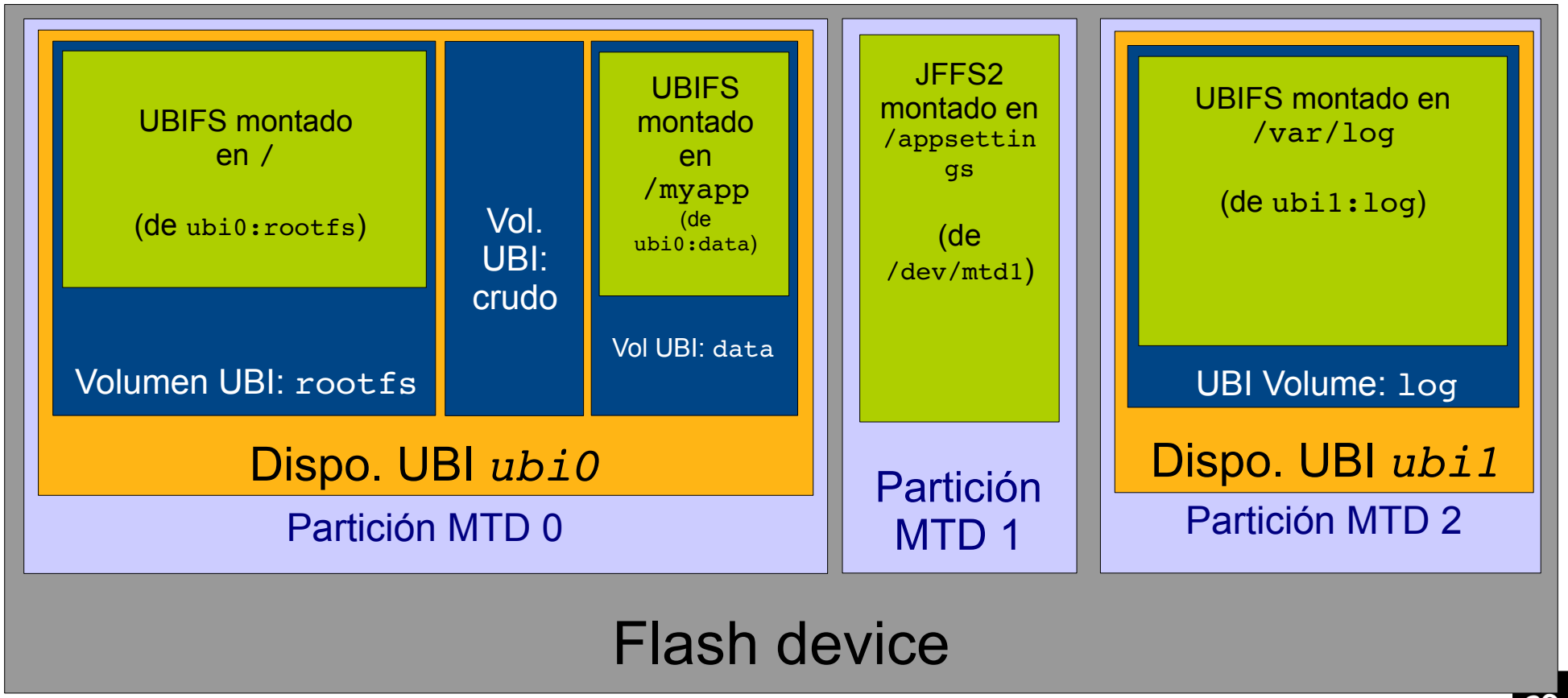
UBIFS

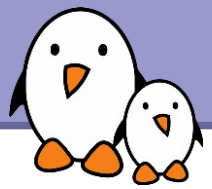
http://www.linux-mtd.infradead.org/doc/ubifs.html

- La siguiente generación de jffs2, de los mismos programadores de linux-mtd.
- En Linux desde 2.6.27
- Trabaja sobre volúmenes UBI.
- Tiene mucho overhead de metadata para particiones chicas. (4M, 8M)



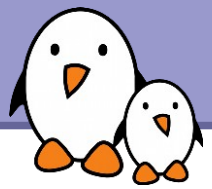
Layout UBI


UBIFS montado en /
(de ubi0:rootfs)
Volumen UBI: rootfs
Vol. UBI: crudo
UBIFS montado en /myapp
(de ubi0:data)
Vol UBI: data
Dispo. UBI ubi0
Partición MTD 0
JFFS2 montado en /appsettings
(de /dev/mtd1)
Partición MTD 1
UBIFS montado en /var/log
(de ubi1:log)
UBI Volume: log
Dispo. UBI ubi1
Partición MTD 2
Flash device



UBI – Características principales

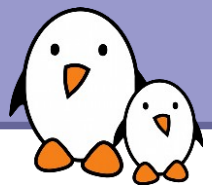
- Escala: El tiempo de montaje, uso de memoria y velocidad de acceso casi no depende del tamaño de la memoria.
- Monta velozmente: No necesita escanear todo el dispositivo al momento de montar.
- Write-back: Mejora notoriamente el desempeño del sistema.
- Tolerante a cortes: Al ser un fs con journal no se corrompe por “apagadas” repentinas.
- Agrega mucho overhead en memoria. Lo hace casi imposible de usar con pocos megas de flash.



SquashFS

http://squashfs.sourceforge.net/

- Filesystem para dispositivos de bloque. No soporta la API MTD.
- Sin embargo, como es read only, funciona bien con `mtdblock`, siempre y cuando el chip no tenga sectores defectuosos.
- Se puede usar para partes readonly del filesystem. Pero no te podes confiar en él. Siempre pueden aparecer los sectores defectuosos.



Conclusiones

- Convertir las particiones jffs2 a ubifs!
- A menos que tengas menos de 10MB de memoria, donde UBI pierde mucho lugar.
- Se puede usar SquashFS para meter todavía más datos en la memoria flash. Recomendable usarlo sobre UBI, para que todos los sectores participen del wear leveling y bad blocks.

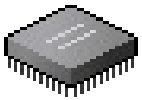
SquashFS

MTD block

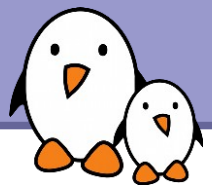
MTD API

UBI

MTD driver

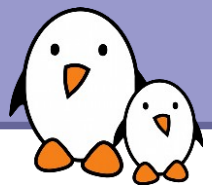


Flash chip



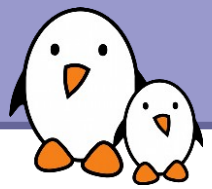
Almacenamiento flash-block

- La flash no se puede acceder, sólo mediante la interfaz de bloque.
- Por lo tanto, no hay manera de acceder a la interfaz de bajo nivel de la memoria y usar los FS de Linux para hacer wear leveling.
- No hay detalles de la capa (Flash Translation Layer) que usan. Éstos se guardan como secretos del fabricante y pueden esconder comportamientos pobres, o mala calidad.
- Por lo tanto, es muy recomendable limitar la cantidad de escrituras en estos dispositivos.



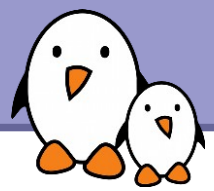
Reduciendo las escrituras

- Por supuesto, no usar flash para swap (igualmente es raro tener swap en embebidos)
- Montar read only o usar read only FSs (SquashFS) siempre que sea posible
- Mantener archivos volátiles en RAM (tmpfs)
- Usar la opción de montaje `noatime`, para evitar actualizar el filesystem cada vez que se lea un archivo. Si se necesita saber el tiempo de acceso, usar la opción `relatime` (por omisión desde Linux 2.6.30).
- No usar la opción de montaje `sync` (escribir inmediatamente). Usar la llamada a sistema `fsync()` para forzar sincronización por archivos.
- Podes no usar FSs con journal. Causan más escrituras, pero son más robustos (trade-off).

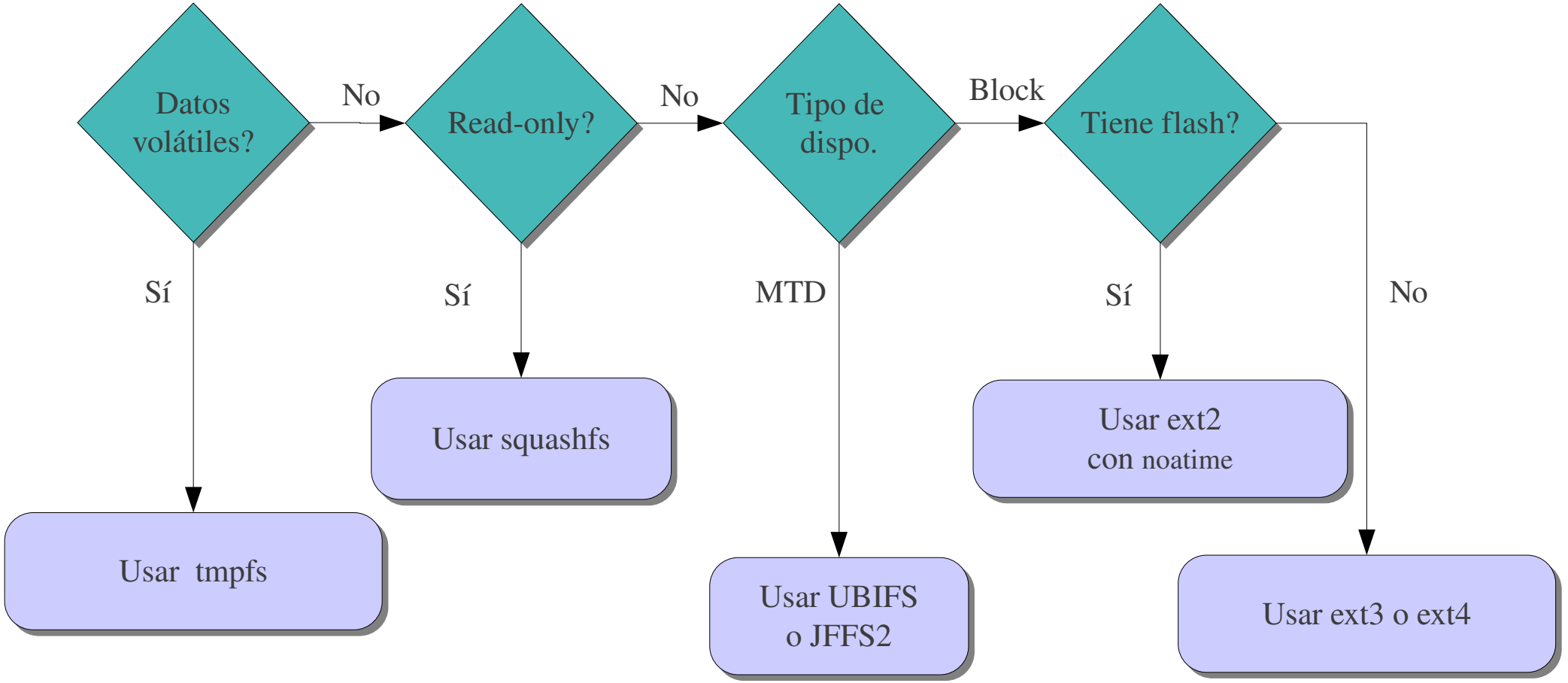


Lecturas recomendadas

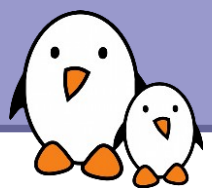
- Introducción a JFFS2 y LogFS:
  http://lwn.net/Articles/234441/
- Linda presentación sobre UBI:
  http://free-electrons.com/redirect/celf-ubi.html
- Documentación de linux-mtd:
  http://www.linux-mtd.infradead.org/



Resumiendo...

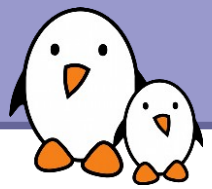


Ver la documentación (con detalles) de todos los FSs del Kernel en `Documentation/filesystems/`.



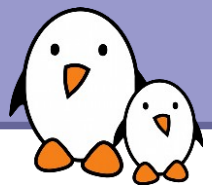
Práctica – Sistemas flash

- Crear particiones en los dispositivos flash.
- Formatear la partición principal en SquashFS sobre mtdblock.
- Usar jffs2 para los datos.



GNU / Linux desktop

Emuladores



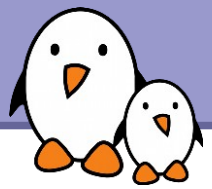
qemu

http://fabrice.bellard.free.fr/qemu/
Emulador de procesador.

QEMU

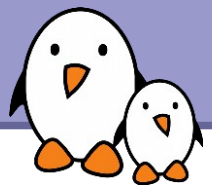
Emulador de sistema completo

- Emula el procesador y varios periféricos
  Supporta: `x86`, `x86_64`, `ppc`, `arm`, `sparc`, `mips`, `m68k`
- Para saber que archs. están soportadas:
  `qemu-system-arm -M ?`
- Emulación de `i386`, `x86_64`: cercana a la velocidad nativa gracias al módulo de Kernel `kqemu` (ahora GPL v2!).



Otros emuladores

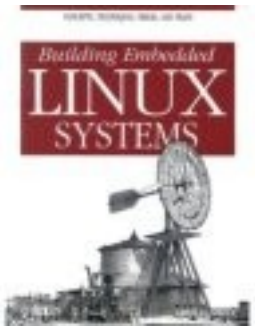
- ARM
  - SkyEye: http://skyeye.sourceforge.net
    Emula varias plataformas ARM (AT91, Xscale...) y puede bootear varios sistemas operativos. (Linux, uClinux, otros...)
  - Softgun: http://softgun.sourceforge.net
    Sistema ARM virtual con muchos periféricos on-board.
    Corre Linux.
  - SWARM - Software ARM - emulador arm7
    http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html
    Puede correr uClinux
- ColdFire emulator
  http://www.slicer.ca/coldfire/
  Puede correr uClinux



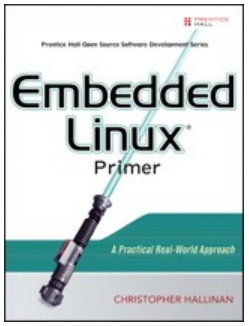
Referencias

Lecturas útiles

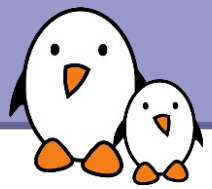
- Embedded Linux System Design and Development
  P. Raghavan, A. Lad, S. Neelakandan, Auerbach, Dec. 2005.
  http://free-electrons.com/redirect/elsdd-book.html



- Building Embedded Linux Systems, O'Reilly
  By Karim Yaghmour, Jon Masters, Gilad Ben-Yossef and Philippe Gerum, and others (including Michael Opdenacker), August 2008
  http://oreilly.com/catalog/9780596529680/



- Embedded Linux Primer, Prentice Hall
  By Christopher Hallinan, September 2006



Sitios útiles

LinuxDevices.com: http://linuxdevices.com

- Notas semanales con noticias y anuncios de dispositivos que usan Linux.
- Artículos, notas técnicas y catálogo de productos.
- Una página excelente para seguir los desarrollos de la industria.