



Embedded Linux system development

Free Electrons

© Copyright 2022, Luciano Diamand.

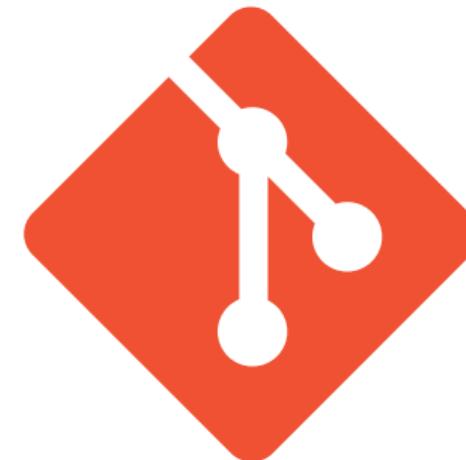
Creative Commons BY-SA 3.0 license.

Última actualización: March 22, 2022.

Actualizaciones del documento y fuentes:

<http://free-electrons.com/doc/training/embedded-linux>

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Derechos de copia

© Copyright 2022, Luciano Diamand

Licencia: Creative Commons Attribution - Share Alike 3.0

<https://creativecommons.org/licenses/by-sa/3.0/legalcode>

Ud es libre de:

- ▶ copiar, distribuir, mostrar y realizar el trabajo
- ▶ hacer trabajos derivados
- ▶ hacer uso comercial del trabajo

Bajo las siguientes condiciones:

- ▶ **Atribución.** Debes darle el crédito al autor original.
- ▶ **Compartir por igual.** Si altera, transforma o construye sobre este trabajo, usted puede distribuir el trabajo resultante solamente bajo una licencia idéntica a ésta.
- ▶ Para cualquier reutilización o distribución, debe dejar claro a otros los términos de la licencia de este trabajo.
- ▶ Se puede renunciar a cualquiera de estas condiciones si usted consigue el permiso del titular de los derechos de autor.

El uso justo y otros derechos no se ven afectados por lo anterior.

Document sources: <https://github.com/bootlin/training-materials/>



Información sobre el curso

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!

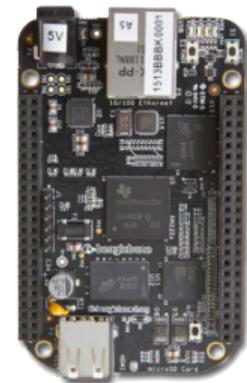




Hardware utilizado en esta sesión de entrenamiento

BeagleBone Black o BeagleBone Black Wireless de BeagleBoard.org

- ▶ Texas Instruments AM335x (CPU ARM Cortex-A8)
- ▶ SoC con aceleración 3D, procesadores adicionales de tiempo real (PRUs) y muchos periféricos.
- ▶ 512 MB de RAM
- ▶ 2 GB de almacenamiento eMMC en la placa (4 GB en la Rev C)
- ▶ puertos USB host y USB device, conector microSD, micro HDMI
- ▶ WiFi y Bluetooth (en la versión wireless), y Ethernet
- ▶ peines de 2 x 46 pins, con acceso a varios buses de expansión (I2C, SPI, UART y más)
- ▶ Un inmenso número de placas de expansión, llamadas *capes*. Ver https://elinux.org/Beagleboard:BeagleBone_Capes.





Participe!

Durante las lecturas...

- ▶ No vacile en hacer preguntas. Otras personas en la audiencia pueden tener preguntas similares también.
- ▶ Esto ayuda al entrenador a detectar cualquier explicación que no haya sido clara o no haya sido profundizada.
- ▶ No vacile en compartir su experiencia, por ejemplo, comparando Linux/Android con otros sistemas operativos que se utilicen en su compañía.
- ▶ Su punto de vista es muy valioso, porque puede ser similar al de sus colegas y diferente al del entrenador.
- ▶ Su participación puede hacer nuestra sesión más interactiva y que los temas sean más fáciles de aprender.



Durante los laboratorios prácticos...

- ▶ Abra la copia electrónica del material de lectura y úsela en todo el laboratorio práctico para encontrar las diapositivas que necesite nuevamente.
- ▶ No copie y pegue desde las diapositivas PDF.
Las diapositivas contienen caracteres UTF-8 que se ven igual a los caracteres ASCII, pero no van a ser interpretados por los shells o los compiladores.



Como en la comunidad de Software Libre y Código abierto, la cooperación durante los laboratorios prácticos es valiosa en esta sesión de entrenamiento:

- ▶ Si ud completa los laboratorios antes que otras personas, no vacile en ayudar a otras personas e investigar problemas que enfrenten. Mientras más rápido progresemos como grupo, más tiempo vamos a tener para explorar temas adicionales.
- ▶ Explique lo que entendió a otros participantes cuando sea necesario. También ayuda a consolidar sus conocimientos.
- ▶ No dude en reportar posibles errores a su instructor.
- ▶ No dude en buscar también soluciones en Internet.



Ayuda memoria de comandos

- ▶ Esta ayuda memoria proporciona ejemplos de necesidades típicas (buscar archivos, extraer un archivo tar ...)
- ▶ Nos ahorra 1 día de formación en línea de comandos UNIX / Linux.
- ▶ Nuestro mejor consejo: en el shell de línea de comandos, siempre presione la tecla Tab para completar los nombres de los comandos y las rutas de archivo. Esto evita el 95% de errores de escritura.
- ▶ Obtenga una copia electrónica en
https://bootlin.com/doc/legacy/command-line/command_memento.pdf





Comandos básicos del vi

- ▶ El editor `vi` es muy útil para hacer cambios rápidos en archivos en un destino embebido
- ▶ Aunque no es muy fácil de usar al principio, `vi` es muy potente y sus 15 comandos principales son fáciles de aprender y son suficientes para el 99% de las necesidades de todos!
- ▶ Obtenga una copia electrónica en https://bootlin.com/doc/legacy/command-line/vi_memento.pdf
- ▶ También puede realizar el tutorial rápido ejecutando `vimtutor`. Esta es una inversión digna!

The screenshot shows the first few pages of the 'vi basic commands' manual page. It includes sections like 'Entering command mode', 'Moving the cursor', 'Replacing strings', and 'Applying a command several times - Examples'. The text is in English. At the bottom right, there is a cartoon penguin icon with the text 'Free Documentation Project'.

vi basic commands

Summary of most useful commands

Entering command mode

Moving the cursor

Replacing strings

Applying a command several times - Examples

Exiting and saving



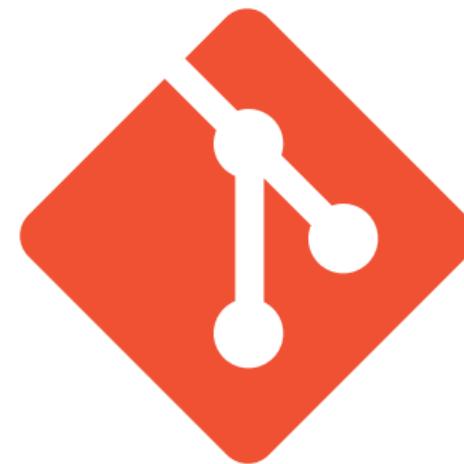
Introducción a Linux Embebido

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Nacimiento del software libre

- ▶ 1983, Richard Stallman, **proyecto GNU** y el concepto de **software libre**. Comienza el desarrollo de *gcc*, *gdb*, *glibc* y otras herramientas importantes
- ▶ 1991, Linus Torvalds, **proyecto Linux kernel**, un nucleo de sistema operativo similar a Unix. Junto con el software GNU y otros componentes de código abierto: forman un sistema operativo completo GNU/Linux
- ▶ 1995, Linux es más popular en sistemas servidor
- ▶ 2000, Linux es más popular en **sistemas embebidos**
- ▶ 2008, Linux es más popular en dispositivos móviles
- ▶ 2010, Linux es más popular en teléfonos



¿Software libre?

- ▶ Un programa es considerado **libre** cuando su licencia ofrece a todos sus usuarios las siguientes **cuatro** libertades
 - ▶ Libertad de ejecutar el Software para cualquier propósito
 - ▶ Libertad de estudiar el Software y modificarlo
 - ▶ Libertad de redistribuir copias
 - ▶ Libertad de distribuir copias de versiones modificadas
- ▶ Estas libertades están concedidas para uso tanto comercial como no-comercial
- ▶ Implican la disponibilidad del código fuente, el Software puede ser modificado y distribuido a los clientes
- ▶ **Una opción interesante para los sistemas embebidos!**



Linux embebido es el uso del **kernel de Linux** y varios componentes **open-source** en sistemas embebidos



Ventajas de Linux y open-source para sistemas embebidos



Reutilización de componentes

- ▶ La principal ventaja de Linux y open-source en sistemas embebidos es la **habilidad** de reutilizar componentes
- ▶ El ecosistema de open-source ya provee de varios componentes para características estandares, desde soporte de Hardware hasta protocolos, pasando por multimedia, gráficos, bibliotecas criptográficas, etc.
- ▶ Tan pronto como un dispositivo Hardware, un protocolo, o una característica se torna conocida, existen varias posibilidades de tener un componente open-source que lo soporte.
- ▶ Permite diseñar de forma rápida y desarrollar productos complejos, basados en componentes existentes.
- ▶ No es necesario redesarrollar otro kernel del sistema operativo, una pila TCP/IP, una pila USB u otra biblioteca gráfica.
- ▶ **Permite enfocarse en el valor agregado del producto.**



- ▶ El Software libre puede ser duplicado en la cantidad de dispositivos que se quiera, libre de cargos.
- ▶ Si su sistema embebido utiliza solo Software libre, se puede reducir los costos de licencias de Software a cero. Incluso, las herramientas de desarrollo son libres, a menos que se elija una versión de Linux embebido comercial.
- ▶ **Permite tener un mayor presupuesto para el Hardware o para incrementar las habilidades y conocimiento de la compañía**



- ▶ Con código abierto, disponemos del código fuente de todos los componentes del sistema
- ▶ Permite modificaciones ilimitadas, cambios, ajustes, depuración, optimización, por un período de tiempo ilimitado
- ▶ Sin una dependencia "bloqueante" de un vendedor externo
 - ▶ Componentes que no sean de código abierto se deben evitar cuando el sistema se diseña y desarrolla
- ▶ **Permite tener un control total sobre el Software que forma parte del sistema**



- ▶ Varios componentes de código abierto son ampliamente utilizados en millones de sistemas
- ▶ Generalmente son de mayor calidad que los desarrollos in-house o incluso de los vendedores propietarios
- ▶ Por supuesto, no todos los componentes de código abierto son de buena calidad, pero los más ampliamente utilizados los son.
- ▶ **Permite diseñar su sistema con componentes fundacionales de alta calidad**



Facilita la prueba de nuevas características

- ▶ Dada la disponibilidad del código abierto, es simple obtener una copia del Software para evaluarlo
- ▶ Permite de forma simple estudiar las opciones mientras se está decidiendo
- ▶ Mucho más simple que la compra y procesos de demostración necesarios en la mayoría de los productos propietarios
- ▶ **Permiten explorar de forma simple nuevas posibilidades y soluciones**



- ▶ Los componentes de Software de código abierto son desarrollados por comunidades de desarrolladores y usuarios
- ▶ Esta comunidad puede proveer soporte de alta calidad: se puede contactar directamente a los desarrolladores principales del componente que se está usando. La probabilidad de obtener una respuesta no depende de la compañía para la que trabajemos.
- ▶ En general mejor que el soporte tradicional, pero es necesario entender como funciona la comunidad para hacer un uso correcto de las posibilidades de soporte
- ▶ **Permite acelerar la resolución de problemas cuando se esté desarrollando el sistema**



- ▶ La posibilidad de formar parte de la comunidad de desarrollo de algunos de los componentes utilizados en sistemas embebidos: reporte de fallas, prueba de nuevas versiones y características, parches que corrigen errores o agregan nuevas características, etc.
- ▶ La mayoría del tiempo, los componentes de código abierto no son el núcleo de valor del producto: es interés de todos contribuir al mismo
- ▶ Para los *ingenieros*: una forma muy **motivante** de ser reconocido fuera de la compañía, comunicarse con otros en el mismo campo, **la oportunidad de nuevas posibilidades**, etc.
- ▶ Para los *gerentes*: **factor de motivación** para los ingenieros, permite a la compañía ser **reconocida** en la comunidad de código abierto y por lo tanto obtener soporte de forma más simple y ser **más atractivo** para los desarrolladores de código abierto



Algunos ejemplos de sistemas embebidos ejecutando Linux



Ruters personales





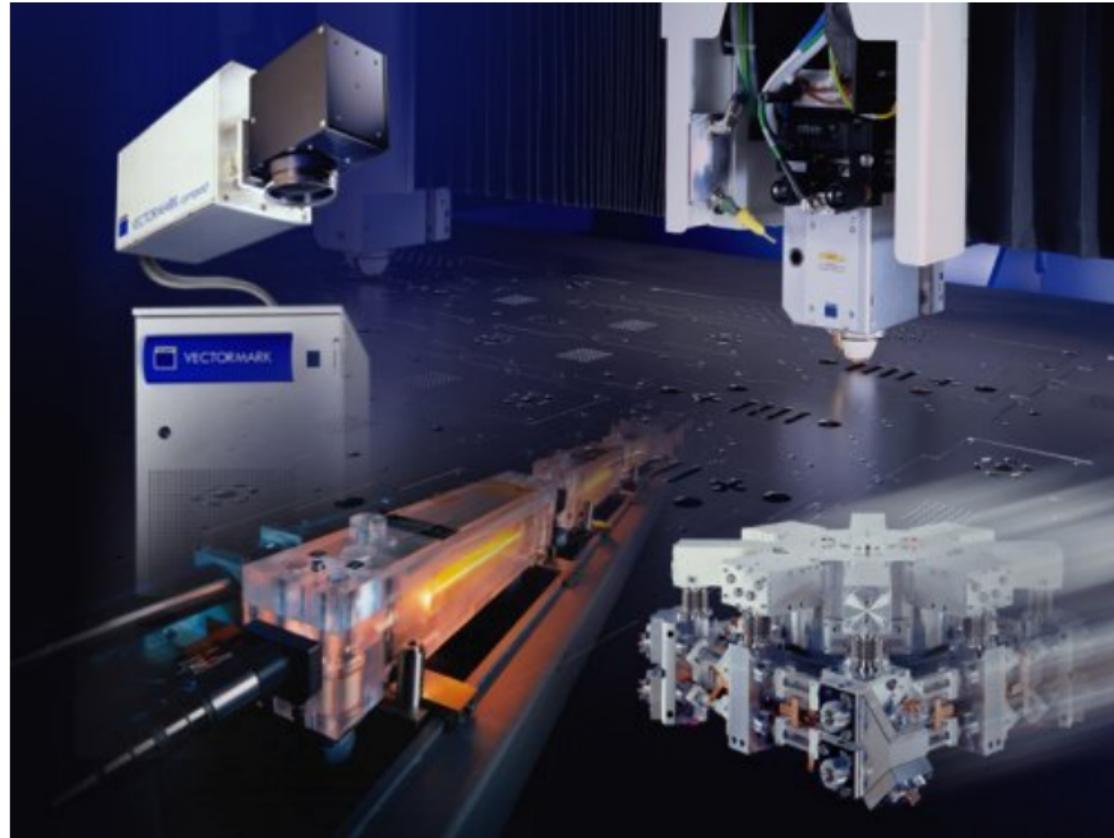


Terminales de punto de venta





Maquinas de corte Laser





Maquina de Viticultura





Hardware para sistemas embebidos en Linux



- ▶ El kernel de Linux y la mayoría de los componentes dependientes de la arquitectura soportan un amplio rango de arquitecturas de 32 y 64 bits
 - ▶ x86 y x86-64, como se encuentran en plataformas PC, pero también sistemas embebidos (multimedia, industrial)
 - ▶ ARM, con miles de diferentes SoC (multimedia, industrial)
 - ▶ PowerPC (principalmente aplicaciones en tiempo real e industriales)
 - ▶ MIPS (principalmente para aplicaciones de red)
 - ▶ SuperH (principalmente aplicaciones multimedia)
 - ▶ Blackfin (arquitectura DSP)
 - ▶ Microblaze (soft-core para FPGA de Xilinx)
 - ▶ Coldfire, SCore, Tile, Xtensa, Cris, FRV, AVR32, M32R



- ▶ Ambas arquitecturas, MMU y no-MMU son soportadas, aunque la arquitectura no-MMU tiene algunas limitaciones.
- ▶ Linux no está diseñado para microcontroladores pequeños.
- ▶ Salvo el juego de herramientas, el gestor de arranque y el kernel, todos los otros componentes son generalmente **independientes de la arquitectura**



- ▶ **RAM:** un sistema básico con Linux puede funcionar con 8 MB de RAM, pero un sistema más realista usualmente requiere 32 MB de RAM. Depende del tipo y tamaño de la aplicación.
- ▶ **Almacenamiento:** un sistema básico con Linux puede trabajar con 4 MB de almacenamiento, pero usualmente se requiere una mayor cantidad.
 - ▶ El almacenamiento en Flash está soportado, tanto flash NAND como NOR, con sistemas de archivos específicos
 - ▶ Almacenamiento en bloque incluyendo tarjetas SD/MMC y eMMC son soportadas
- ▶ Es preferible no tener muchas restricciones en la cantidad de RAM/almacenamiento: tener una cierta flexibilidad en este nivel nos permite reutilizar tantas aplicaciones como sea posible.



- ▶ El kernel de Linux tiene soporte para varios protocolos de comunicaciones comunes
 - ▶ I2C
 - ▶ SPI
 - ▶ CAN
 - ▶ 1-wire
 - ▶ SDIO
 - ▶ USB
- ▶ Y también soporte para redes extensivo
 - ▶ Ethernet, Wifi, Bluetooth, CAN, etc.
 - ▶ IPv4, IPv6, TCP, UDP, SCTP, DCCP, etc.
 - ▶ Firewalling, ruteo avanzado, multicast



- ▶ **Plataformas de evaluación** del proveedor de SoC. Usualmente costosas, pero con muchos perifericos incluidos. Generalmente no recomendable para productos reales.
- ▶ **Componente en Módulo**, una pequeña placa solamente con CPU/RAM/flash y algunos otros componentes, con conectores para acceder a todos los periféricos. Puede ser utilizado para construir productos finales en pequeñas o medianas cantidades.
- ▶ **Plataformas de desarrollo comunitario**, una nueva tendencia para hacer un SoC particular popular y facilmente disponible. Estas están listas para utilizar y a un bajo costo, pero generalmente tienen menos periféricos que las plataformas de evaluación. En algunos casos pueden ser utilizadas para productos reales.



- ▶ **Plataforma personalizada.** Los esquemáticos para placas de evaluación o plataformas de desarrollo están disponibles más comúnmente de forma libre, haciendo mas simple el desarrollo de plataformas personalizadas.



- ▶ Aseguresé que el Hardware que planea utilizar está actualmente soportado por el kernel de Linux, y posee un gestor de arranque de código abierto, especialmente para el SoC destino.
- ▶ Teniendo soporte en las versiones oficiales de los proyectos (kernel, gestor de arranque): la calidad es mejor, y están disponibles nuevas versiones.
- ▶ Algunos proveedores de SoC y/o proveedores de placas no contribuyen sus cambios hacia la línea principal del kernel de Linux. Recomiende que lo hagan, o utilice otro producto de ser posible. Una buena métrica es ver las diferencias entre su kernel y el oficial.
- ▶ **Entre un hardware correctamente soportado en el kernel oficial de Linux y un hardware mal soportado, existirán enormes diferencias en tiempo de desarrollo y costo.**



Arquitectura de sistemas embebidos en Linux



PC de desarrollo

Herramientas
compilador
depurador
...

Sistema embebido

Aplicación

Aplicación

Biblioteca

Biblioteca

Biblioteca

Biblioteca

Biblioteca C

Linux kernel

Bootloader



- ▶ Juego de herramientas de compilación cruzada
 - ▶ Compilador que corre en la máquina de desarrollo, pero genera código para el destino
- ▶ Gestor de arranque
 - ▶ Ejecutado por el Hardware, responsable de la inicialización básica, carga y ejecuta el kernel
- ▶ Kernel de Linux
 - ▶ Contiene los procesos y el manejo de memoria, red, controladores de dispositivos y provee servicios para la aplicación en el espacio de usuario
- ▶ Biblioteca C
 - ▶ La interfaz entre el kernel y las aplicaciones en el espacio de usuario
- ▶ Bibliotecas y aplicaciones
 - ▶ De terceros (Third-party or in-house)



Varias tareas de distinto tipo se necesitan al momento de desplegar Linux embebido en un producto:

- ▶ **Board Support Package development (BSP)**
 - ▶ Un BSP contiene un gestor de arranque y un kernel con los device drivers adecuados para el Hardware destino
 - ▶ Es el propósito del entrenamiento en *desarrollo del Kernel*
- ▶ **Integración del sistema**
 - ▶ Integrar todos los componentes, gestor de arranque, kernel, bibliotecas de terceros y aplicaciones in-house en un sistema funcional
 - ▶ Es el propósito de este entrenamiento
- ▶ **Desarrollo de aplicaciones**
 - ▶ Aplicaciones Linux normales, pero utilizando bibliotecas especialmente elegidas



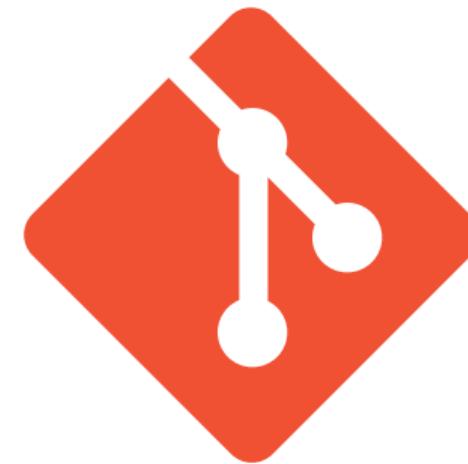
Entorno de desarrollo de Linux Embebido

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!



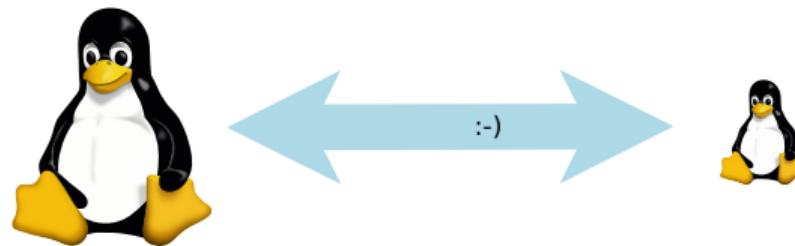


- ▶ Dos formas de cambiar a Linux embebido
 - ▶ Utilice **soluciones proporcionadas y apoyadas por vendedores** como MontaVista, Wind River o TimeSys. Estas soluciones vienen con sus propias herramientas de desarrollo y entorno. Utilizan una mezcla de componentes de código abierto y herramientas propietarias.
 - ▶ Utilice **soluciones comunitarias**. Están completamente abiertas, soportado por la comunidad.
- ▶ En estas sesiones de formación, no promovemos un Proveedor y, por lo tanto, utilizamos soluciones comunitarias
 - ▶ Sin embargo, conociendo los conceptos, cambiar a otro proveedor es simple



SO para el desarrollo de Linux

- ▶ Recomendamos encarecidamente el uso de Linux como sistema para incorporar a los desarrolladores de Linux, por múltiples razones.
- ▶ Todas las herramientas de la comunidad están desarrolladas y diseñadas para funcionar en Linux. Intentar utilizarlos en otros sistemas operativos (Windows, Mac OS X) conducirá a problemas, y su uso en estos sistemas generalmente no son apoyados por los desarrolladores de la comunidad.
- ▶ Como Linux también se ejecuta en el dispositivo embebido, todo el conocimiento ganado por el uso de Linux en el escritorio se aplicará de manera directa al dispositivo embebido.





- ▶ **Cualquier distribución de Linux de escritorio suficientemente reciente** se puede utilizar para la estación de trabajo de desarrollo
 - ▶ Ubuntu, Debian, Fedora, openSUSE, Red Hat, etc.
- ▶ Hemos elegido Ubuntu, ya que es una distribución de Linux de escritorio **ampliamente utilizado y fácil de usar**
- ▶ La configuración de Ubuntu en las computadoras portátiles de formación ha quedado intacta después del proceso de instalación normal. Aprender Linux embebido también consiste en aprender las herramientas necesarias en la estación de trabajo de desarrollo!





Usuarios root y no root de Linux

- ▶ Linux es un sistema operativo multiusuario
 - ▶ El **usuario root es el administrador**, y puede realizar operaciones privilegiadas como: montar sistemas de archivos, configurar la red, crear archivos de dispositivos, cambiar la configuración del sistema, instalar o eliminar software
 - ▶ Todos los **otros usuarios no tienen privilegios**, y no pueden realizar estas operaciones a nivel de administrador
- ▶ En un sistema Ubuntu, no es posible iniciar sesión como root, sólo como usuario normal.
- ▶ El sistema se ha configurado para que la primer cuenta de usuario creada pueda ejecutar operaciones privilegiadas a través de un programa llamado `sudo`.
 - ▶ Ejemplo: `sudo mount /dev/sda2 /mnt/disk`



Paquetes de Software

- ▶ El mecanismo de distribución para el software en GNU / Linux es diferente de la de Windows
- ▶ Las distribuciones de Linux proporcionan una forma central y coherente de instalar, actualizar y eliminar aplicaciones y bibliotecas: **paquetes**
- ▶ Los paquetes contienen archivos de aplicación o bibliotecas y meta-information asociada a los mismos, tal como versión y dependencias
 - ▶ .deb en Debian y Ubuntu, .rpm en Red Hat, Fedora, openSUSE
- ▶ Los paquetes se almacenan en **repositorios**, usualmente en servidores HTTP o FTP
- ▶ Sólo debe utilizar paquetes de repositorios oficiales para su distribución, a menos que sea estrictamente necesario.



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

- ▶ Los repositorios de paquetes se especifican en `/etc/apt/sources.list`
- ▶ Para actualizar la lista de paquetes del repositorio:
`sudo apt-get update`
- ▶ Para encontrar el nombre de un paquete a instalar, lo mejor es usar el motor de búsqueda en <http://packages.debian.org> o en <http://packages.ubuntu.com>. También puede usar:
`apt-cache search <keyword>`



Gestión de paquetes de Software 2/2

- ▶ Para instalar un paquete dado:

```
sudo apt-get install <package>
```

- ▶ Para eliminar un paquete dado:

```
sudo apt-get remove <package>
```

- ▶ Para instalar todas las actualizaciones de paquetes disponibles:

```
sudo apt-get dist-upgrade
```

- ▶ Para obtener información acerca de un paquete:

```
apt-cache show <package>
```

- ▶ Interfaces gráficas

- ▶ Para GNOME Synaptic

- ▶ Para KDE KPackageKit

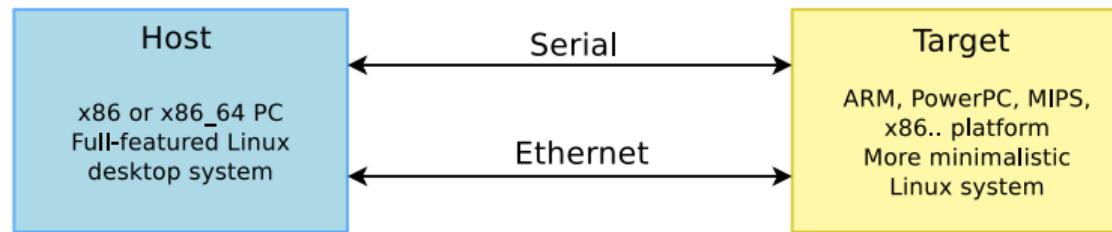
Más detalles sobre la gestión de paquetes:

<http://www.debian.org/doc/manuals/apt-howto/>



Host vs. destino

- ▶ Al trabajar el desarrollo embebido, siempre hay una división entre
 - ▶ El *host*, la estación de trabajo de desarrollo, que es típicamente una PC potente
 - ▶ El *destino*, que es el sistema embebido bajo desarrollo
- ▶ Están conectados por diversos medios: casi siempre una línea serie para fines de depuración, con frecuencia una conexión Ethernet, a veces una interfaz JTAG para la depuración a bajo nivel





Programa de comunicación por línea serie

- ▶ Una herramienta esencial para el desarrollo embebido es un programa de comunicación por línea serie, como HyperTerminal en Windows
- ▶ Hay múltiples opciones disponibles en Linux: Minicom, Picocom, Gtkterm, Putty, etc.
- ▶ En estas sesiones de entrenamiento, recomendamos utilizar el más simple de ellos: **picocom**
 - ▶ Se instala con `sudo apt-get install picocom`
 - ▶ Se ejecuta con `picocom -b BAUD_RATE /dev/SERIAL_DEVICE`
 - ▶ Se termina con Control-A Control-X
- ▶ Donde `SERIAL_DEVICE` generalmente es
 - ▶ `ttyUSBx` para conversores USB a serie
 - ▶ `ttySx` para puertos series reales



Consejos sobre la línea de comandos

- ▶ El uso de la línea de comandos es obligatorio para muchas operaciones necesarias para el desarrollo de Linux
- ▶ Es una manera muy poderosa de interactuar con el sistema, con lo cual usted puede ahorrar mucho tiempo
- ▶ Algunos consejos útiles
 - ▶ Puede utilizar varias pestañas en el Gnome Terminal
 - ▶ Recuerde que puede utilizar rutas de acceso relativas (por ejemplo: ../../linux) además de rutas absolutas (por ejemplo: /home/user)
 - ▶ En un shell, pulse [Control] [r] y, a continuación, una palabra clave, esto buscará a través del historial de comandos. Presione [Control] [r] nuevamente para buscar hacia atrás en la historia
 - ▶ Puede copiar y pegar rutas directamente desde la terminal con arrastrar y soltar



Hardware utilizado en esta sesión de entrenamiento.

Raspberry PI 3 Modelo B

- ▶ CPU Quad Core 1.2GHz Broadcom BCM2837 de 64 bits
- ▶ 1GB RAM
- ▶ BCM43438 wireless LAN y Bluetooth Low Energy (BLE) en la placa
- ▶ 100 Base Ethernet
- ▶ 40-pin extended GPIO
- ▶ 4 puertos USB 2
- ▶ Salida estereo y puerto de video compuesto
- ▶ HDMI de tamaño máximo
- ▶ Puerto de cámara CSI para conectar una cámara Raspberry Pi
- ▶ Puerto Micro SD para cargar el SO y almacenar datos
- ▶ También incluye extensiones a través de HATs
<https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/>





Preparar el entorno del laboratorio

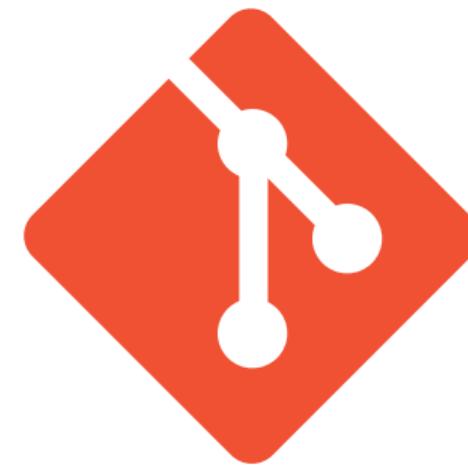
- ▶ Crear la estructura de directorios necesaria
- ▶ Crear el shell-script para la asignación de variables de entorno



Juego de herramientas de compilación cruzada

Juego de herramientas de compilación cruzada

Free Electrons



© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!



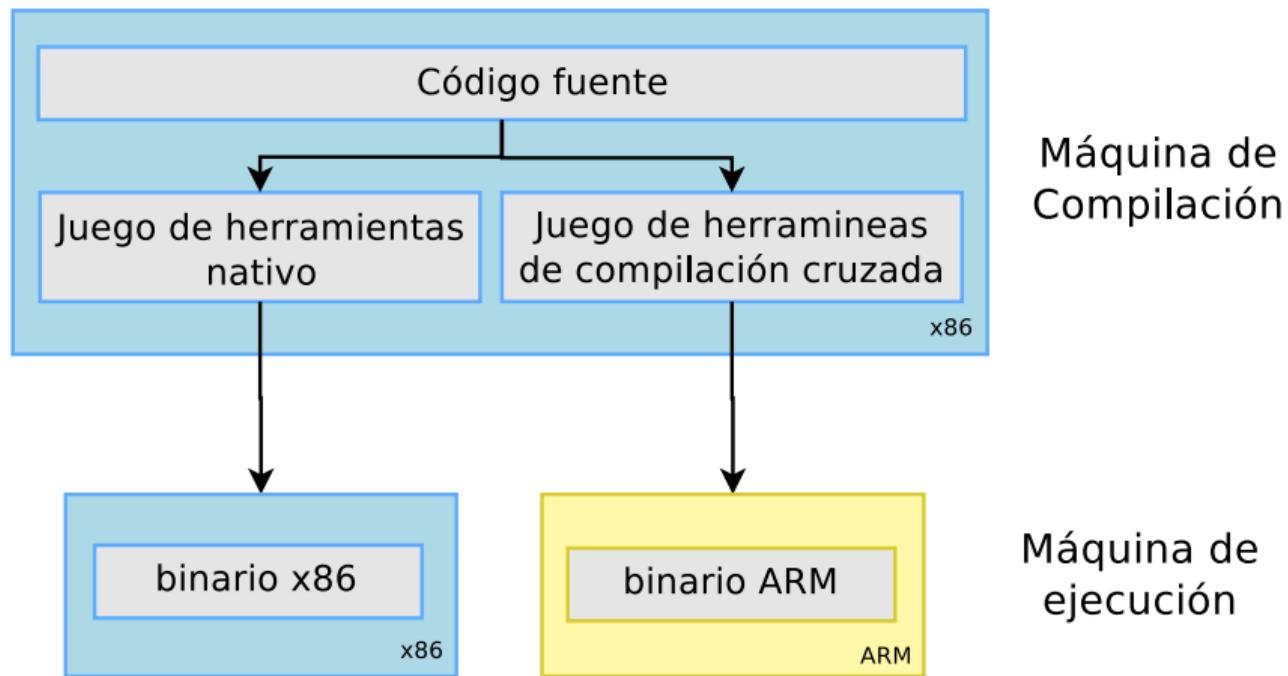
Definiciones y Componentes



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



Definiciones 2/2





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



Construcción del juego de herramientas



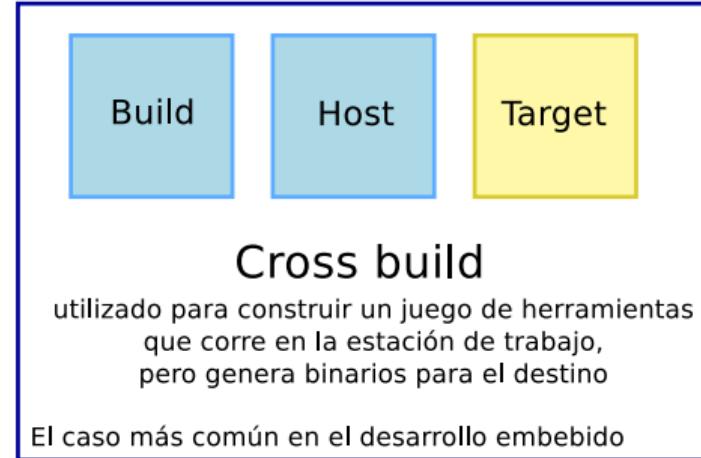
Native build

utilizado para construir el gcc normal de una estación de trabajo



Cross-native build

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



Cross build

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

El caso más común en el desarrollo embebido



Canadian build

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



Binutils

Cabeceras del Kernel

Bibliotecas C/C++

Compilador GCC

Depurador GDB
(opcional)

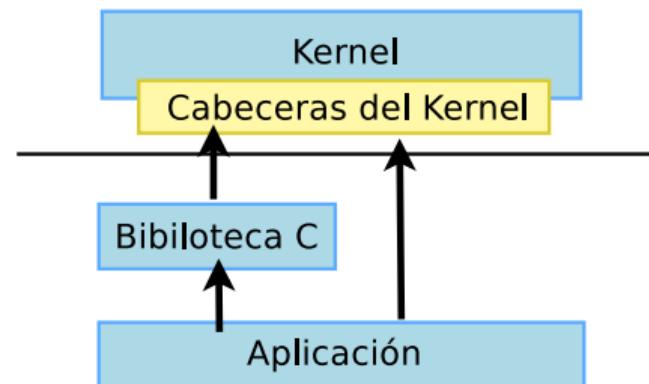
Juego de herramientas de compilación cruzada

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



Cabeceras del Kernel 1/1

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





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

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

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

```
#define O_RDWR 00000002
```

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

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



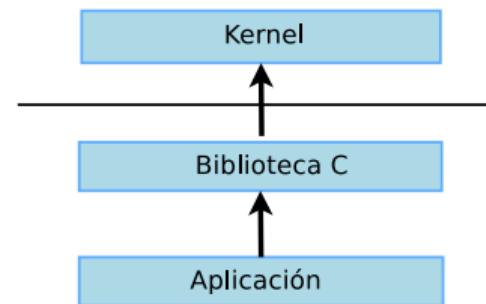
- ▶ El ABI del Kernel al espacio de usuario es **compatible hacia atrás**
 - ▶ Binarios generados con un juego de herramientas utilizando cabeceras del Kernel más viejas que las del Kernel en ejecución van a funcionar sin problemas, pero no se van a poder llamar a las nuevas llamadas al sistema, estructuras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



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



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





Bibliotecas C

- ▶ Licencia: LGPL
- ▶ Biblioteca C del proyecto GNU
- ▶ Diseñada teniendo en cuenta el rendimiento, el cumplimiento de las normas y la portabilidad
- ▶ Se encuentra en todos los sistemas GNU/Linux
- ▶ Por supuesto, mantenido activamente
- ▶ Por defecto, muy grande para los sistemas embebidos pequeños: En armv7hf, versión 2.23: `libc`: 1.5 MB,
`libm`: 492 KB, source:
<https://toolchains.bootlin.com>
- ▶ Sin embargo, algunas funciones que no son necesarias en los sistemas embebidos se pueden configurar (combinadas desde el antiguo proyecto *eglibc*)
- ▶ <http://www.gnu.org/software/libc/>



Imagen: <https://bit.ly/2EzH16m>

- ▶ <http://uclibc-ng.org/>
- ▶ Una continuación del viejo proyecto uClibc, licencia: LGPL
- ▶ Biblioteca de C liviana para sistemas embebidos pequeños
 - ▶ Alta configurabilidad: se pueden activar o deshabilitar muchas funciones a través de una interfaz menuconfig
 - ▶ Admite la mayoría de las arquitecturas embebidas, incluidas las que no tienen MMU (ARM Cortex-M, Blackfin, etc.). La única biblioteca que soporta ARM noMMU
 - ▶ No hay compatibilidad binaria garantizada. Puede necesitar recompilar las aplicaciones cuando cambie la configuración de la biblioteca.
 - ▶ Es posible que algunas funciones de glibc aún no estén implementadas (tiempo real, operaciones de punto flotante ...)
 - ▶ Enfocada en el tamaño más que en el rendimiento
 - ▶ Tamaño en armv7hf, versión 1.0.24: libc: 652 KB, fuente:
<https://toolchains.bootlin.com>
- ▶ Apoyado activamente, pero el Proyecto Yocto dejó de soportarlo.



<http://www.musl-libc.org/>

- ▶ Una biblioteca ligera, rápida y simple para sistemas embebidos
- ▶ Creado mientras el desarrollo de uClibc estaba estancado
- ▶ En particular, genial en hacer pequeños ejecutables estáticos
- ▶ Licencia permisiva (MIT)
- ▶ Soportado por los sistemas de construcción como Buildroot y el proyecto Yocto
- ▶ Utilizada por la distribución de Linux Alpine (<https://www.alpinelinux.org>), que se ajusta a unos 130 MB de almacenamiento.





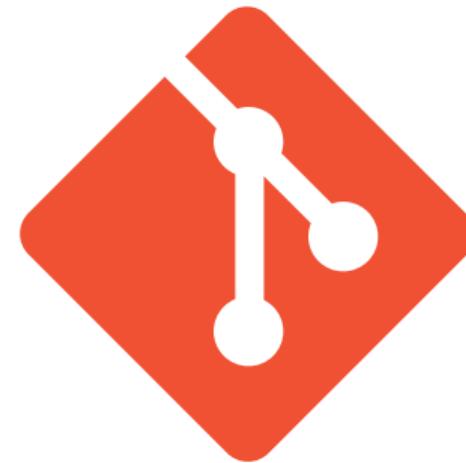
Introducción a Qemu

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!

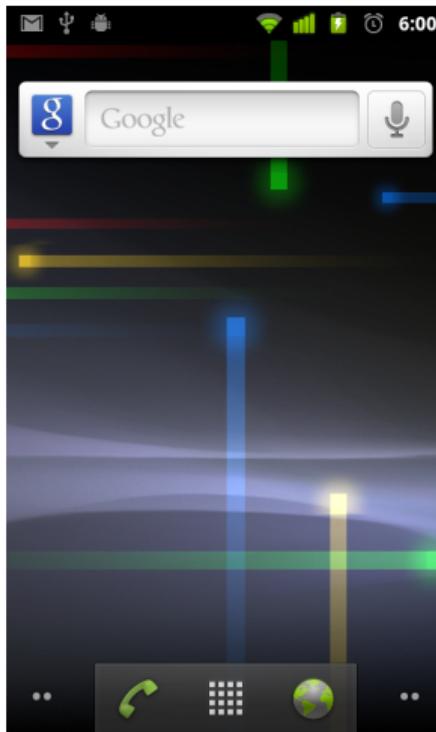




Qemu, Visión General



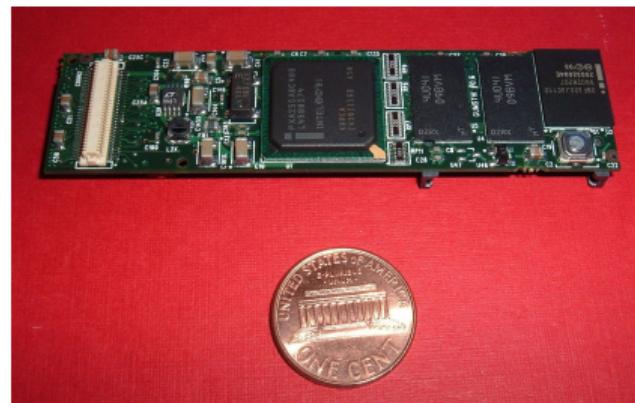
Qemu, Introducción



- ▶ Emula un sistema completo: procesador y periféricos
- ▶ Puede emular varias arquitecturas, incluidas x86, ARM, SPARC, PowerPC
- ▶ Los sistemas operativos como Linux y Windows se pueden ejecutar dentro del sistema emulado
- ▶ Útil para probar y desarrollar sistemas embebidos
- ▶ El emulador de Android es una versión modificada de Qemu

Créditos de la imagen:

https://commons.wikimedia.org/wiki/File:Android_screenshot.png



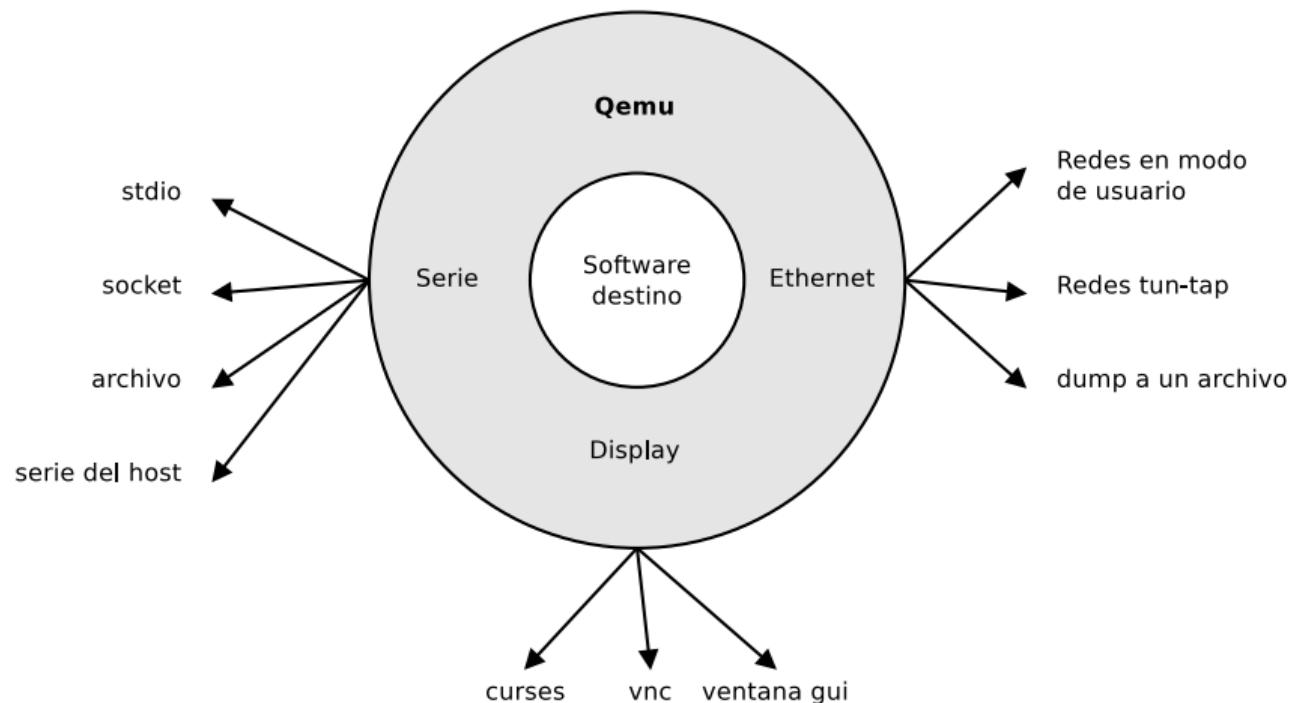
Créditos de la imagen:

<https://commons.wikimedia.org/wiki/File:Gumstix.agr.jpg>

- ▶ Los SoC ARM admitidos incluyen Exynos4210, PXA255, PXA270, OMAP2420, LM3S811
- ▶ La placa Vexpress-a9 basada en PXA270, se utilizará para el desarrollo

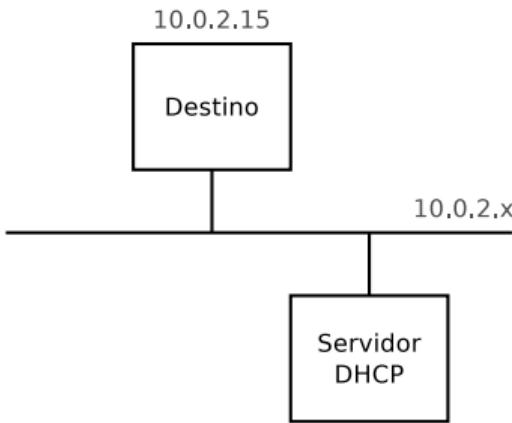


Arquitectura Qemu

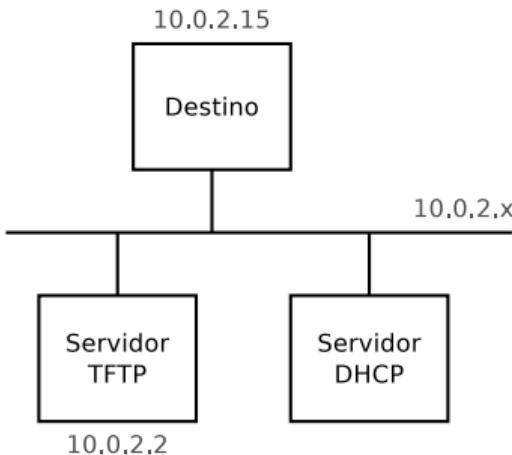




- ▶ Cuando se escribe un paquete en el controlador Ethernet, Qemu analiza la carga útil de IP y UDP/TCP
- ▶ Si el paquete es una conexión TCP, Qemu abre un socket en el sistema host y se conecta a la IP especificada
- ▶ La carga útil de los paquetes TCP posteriores para la conexión, se extrae y se escribe en el socket
- ▶ Lo mismo aplica para los paquetes UDP también



- ▶ Si se recibe un paquete DHCP, Qemu responde con una dirección IP
- ▶ Direcciones IP: 10.0.2.15 - 10.0.2.31



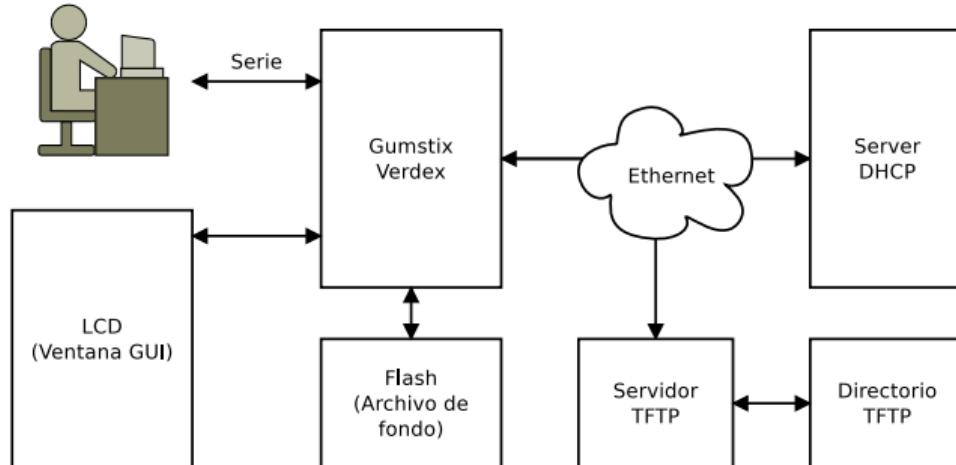
- ▶ TFTP es un protocolo simple basado en UDP para la transferencia de archivos
- ▶ La red en modo de usuario implementa un servidor TFTP
- ▶ Si un paquete UDP está destinado a 10.0.2.2, puerto 69, entonces el servidor TFTP incorporado responde
- ▶ El servidor TFTP de Qemu debe contar con un directorio de donde servir los archivos



Emulación Vexpress



Componentes de la placa Vexpress



- ▶ SDRAM (128 MB)
- ▶ Flash NOR (64 MB)
- ▶ Serial: stdio
- ▶ Ethernet: red en modo de usuario

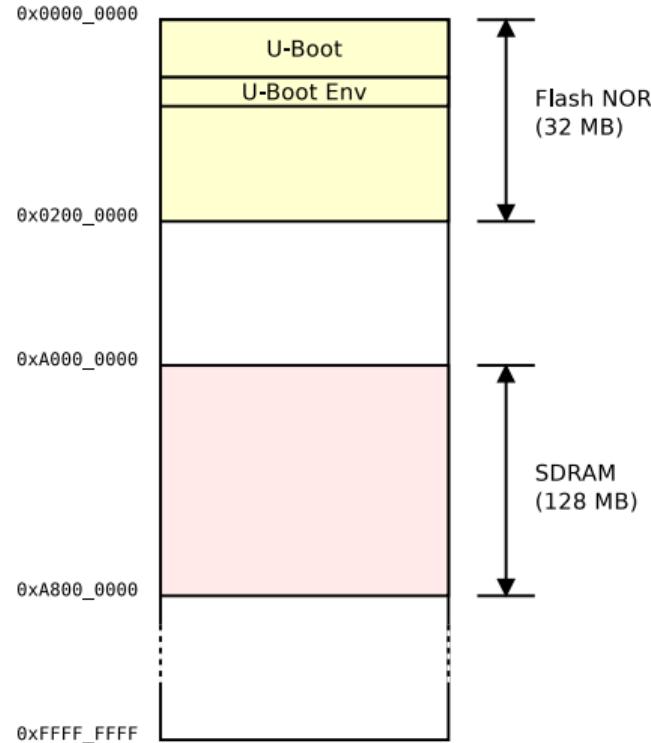


Invocación de Qemu, sin interfaz de usuario Web

```
qemu-system-arm -M vexpress-a9 -pflash flash -m 128M -tftp tftpboot -serial
```



Mapa de memoria





Secuencia de arranque de la placa

- ▶ El procesador PXA270, al reiniciar, comienza a ejecutar instrucciones desde 0x0
- ▶ La Flash NOR está ubicada en la dirección base 0x40000000
- ▶ U-Boot está ubicado en el desplazamiento 0x0 en la Flash NOR
- ▶ Entonces, al reiniciar, el procesador comienza a ejecutar U-Boot
- ▶ U-Boot descarga el núcleo, la Ramdisk y Device Tree Blob a RAM, desde el host, usando TFTP
- ▶ U-Boot transfiere el control al núcleo y proporciona la ubicación del Ramdisk y Device Tree Blob al núcleo



Opciones del Juego de herramientas



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



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



Banderas de optimización de CPU

- ▶ Un juego de herramientas para compilación cruzada es específico para una arquitectura de CPU (ARM, x86, MIPS, PowerPC)
- ▶ Sin embargo, gcc ofrece más opciones:
 - ▶ `-march` permite seleccionar un conjunto de instrucciones específico
 - ▶ `-mtune` permite optimizar código para una CPU específica.
 - ▶ Por ejemplo, `-march=armv7 -mtune=cortex-a8`
 - ▶ `-mcpu=cortex-a8` puede usarse en su lugar para permitir que gcc infiera el conjunto de instrucciones de destino (`-match=armv7`) y las optimizaciones de la CPU (`-mtune=cortex-a8`)
 - ▶ <https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html>
- ▶ En tiempo de compilación del juego de herramientas, se pueden elegir los valores. Son usados:
 - ▶ Como valores por defecto para las herramientas de compilación cruzada, cuando ninguna otra opción `-march`, `-mtune`, `-mcpu` es utilizada
 - ▶ Para compilar la biblioteca C
- ▶ Incluso si la biblioteca C fue compilada para armv5t, no prohíbe de compilar otros programas para armv7



Obteniendo el juego de herramientas



Construir el juego de herramientas de forma manual

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

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



Obtener un juego de herramientas pre-compilado

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



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



Juego de herramientas de Linaro

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





Instalar un juego de herramientas pre-compilado

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



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

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



► **Crosstool-ng**

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



Utilidades para construir el juego de herramientas 3/3

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

▶ **Buildroot**

- ▶ Basado en Makefile. Puede construir juegos de herramientas basado en (e)glibc, uClibc y musl, para una amplia gama de arquitecturas.
- ▶ <http://www.buildroot.net>

▶ **PTXdist**

- ▶ Basado en Makefile, uClibc o glibc, mantenido por *Pengutronix*
- ▶ <http://pengutronix.de/software/ptxdist/>

▶ **OpenEmbedded / Yocto**

- ▶ Muy completo en prestaciones, pero es un sistema de creación más complicado
- ▶ <http://www.openembedded.org/>
- ▶ <https://www.yoctoproject.org/>



Crosstool-NG: instalación y uso

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

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

- ▶ Varias configuraciones para distintas arquitecturas están disponibles a través de ejemplos que se pueden visualizar utilizando

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

- ▶ Para cargar una configuración de ejemplo

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

- ▶ Para ajustar la configuración

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

- ▶ Para construir el juego de herramientas

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



Contenido del juego de herramientas

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



Es momento de construir el juego de herramientas

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



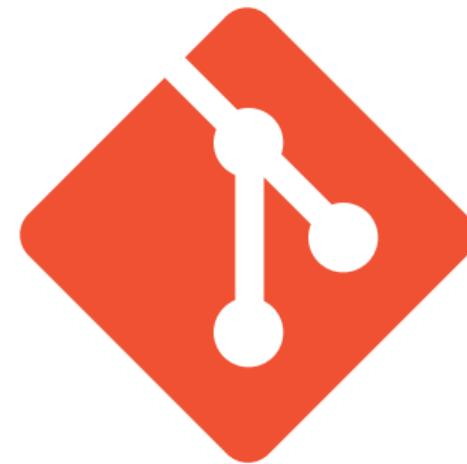
Gestor de arranque

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Secuencia de inicio

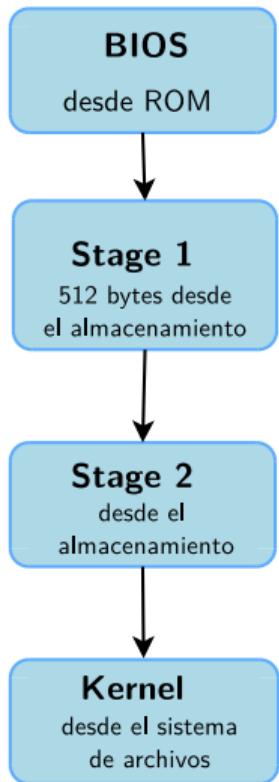


- ▶ El gestor de arranque es la pieza de Software responsable de
 - ▶ Inicializar de forma básica el Hardware
 - ▶ Cargar el binario de una aplicación, generalmente el Kernel del sistema operativo, desde el almacenamiento Flash, desde la red, o desde otro tipo de almacenamiento no volatil.
 - ▶ Posibilita descomprimir el binario de la aplicación
 - ▶ Ejecutar aplicación
- ▶ Además de estas funciones básicas, la mayoría de los gestores de arranque proveen un shell con una serie de comandos implementando diferentes operaciones
 - ▶ Carga de datos desde el almacenamiento o red, inspección de memoria, diagnósticos y pruebas de Hardware, etc.



Gestores de arranque en x86 1/2

- ▶ Los procesadores x86 normalmente se incluyen en una placa con memoria no volátil que contiene un programa, el BIOS.
- ▶ Este programa es ejecutado por CPU luego de reiniciar, y es responsable de la inicialización básica del Hardware y la carga de un fragmento de código desde una memoria no volátil.
 - ▶ Este bloque de código generalmente ocupa los primeros 512 bytes de un dispositivo de almacenamiento
- ▶ Este bloque de código suele ser el gestor de arranque de primer nivel, que cargará el gestor de arranque completo.
- ▶ Típicamente comprende los formatos del sistema de archivos de modo que el archivo del Kernel se puede cargar directamente desde un sistema de archivos normal.



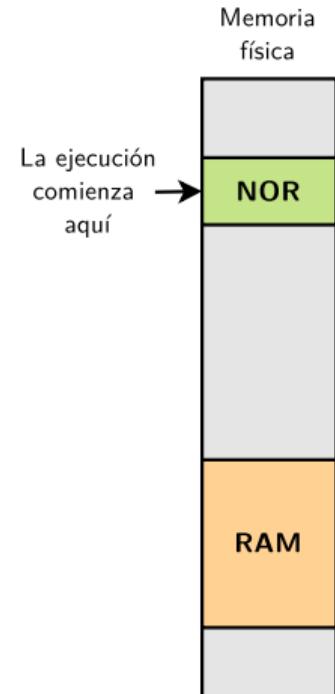


- ▶ GRUB, Grand Unified Bootloader, el más poderoso.
<http://www.gnu.org/software/grub/>
 - ▶ Puede leer muchos formatos de sistema de archivos para cargar la imagen del kernel y la configuración, proporciona una shell poderosa con varios comandos, puede cargar imágenes del kernel a través de la red, etc.
 - ▶ Vea nuestra presentación dedicada para más detalles:
<http://free-electrons.com/docs/grub/>
- ▶ Syslinux, para arranque en red y medios extraíbles (clave USB, CD-ROM)
<http://www.kernel.org/pub/linux/utils/boot/syslinux/>



Arranque en CPUs embebidas: caso 1

- ▶ Cuando se activa, la CPU comienza a ejecutar código en una dirección fija
- ▶ No hay ningún otro mecanismo de arranque proporcionado por la CPU
- ▶ El diseño del hardware debe garantizar que un chip flash NOR esté conectado para que sea accesible en la dirección en la que la CPU comienza a ejecutar instrucciones
- ▶ El gestor de arranque de la primera etapa debe estar programado en esa dirección en la NOR
- ▶ NOR es obligatorio, ya que permite el acceso aleatorio, que NAND no permite
- ▶ **No muy común** (poco práctico, y requiere una Flash NOR)



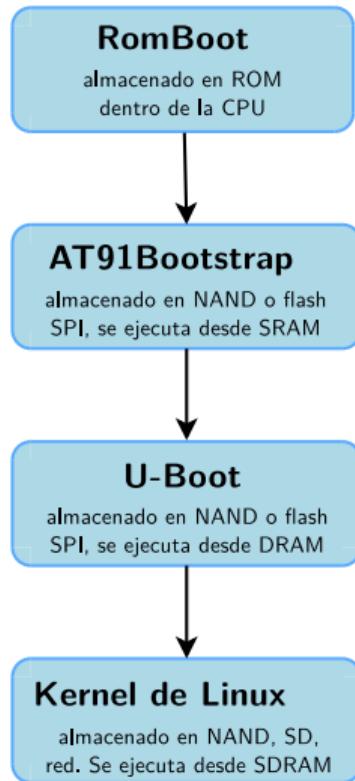


Arranque en CPUs embebidas: caso 2

- ▶ La CPU tiene un código de arranque integrado en ROM
 - ▶ BootROM en CPUs AT91, "código ROM" en OMAP, etc.
 - ▶ Los detalles exactos dependen de la CPU
- ▶ Este código de arranque es capaz de cargar un gestor de arranque de primer nivel desde un almacenamiento en una SRAM interna (la DRAM no se ha inicializado todavía)
 - ▶ El dispositivo de almacenamiento puede ser: MMC, NAND, flash SPI, UART, etc.
- ▶ El gestor de arranque de primer nivel es
 - ▶ Limitado en tamaño debido a restricciones de hardware (tamaño SRAM)
 - ▶ Proporcionado ya sea por el vendedor de la CPU o por proyectos comunitarios
- ▶ Este gestor de arranque de primer nivel debe inicializar la DRAM y otros dispositivos de Hardware y cargar el gestor de arranque de segundo nivel en la RAM



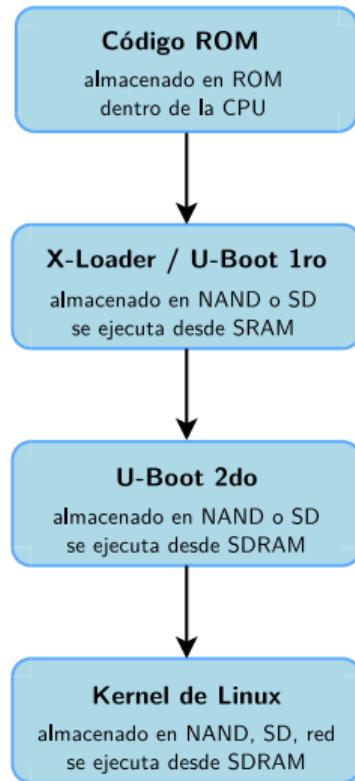
Arranque en ARM Atmel AT91



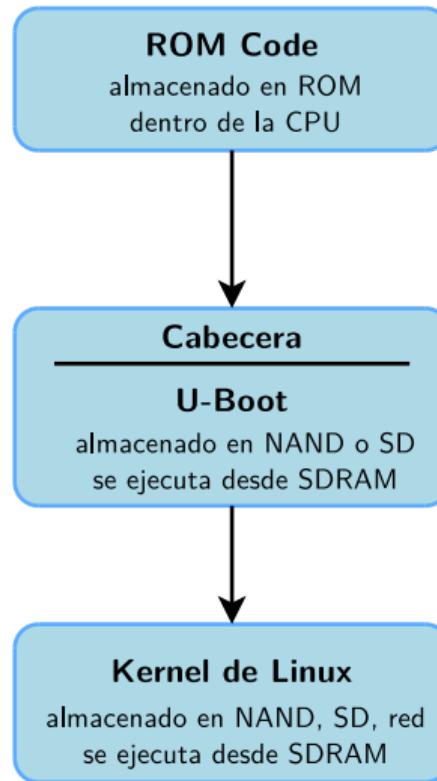
- ▶ **RomBoot:** intenta encontrar una imagen de arranque válida desde varias fuentes de almacenamiento y cargarlo en SRAM (DRAM no inicializada todavía). Tamaño limitado a 4 KB. Ninguna interacción del usuario es posible en el modo de arranque estándar.
- ▶ **AT91Bootstrap:** se ejecuta desde SRAM. Inicializa la DRAM, el controlador NAND o SPI y carga el gestor de arranque secundario en RAM y lo inicia. No es posible la interacción con el usuario.
- ▶ **U-Boot:** se ejecuta desde RAM. Inicializa otros dispositivos hardware (red, USB, etc.). Carga la imagen del núcleo desde almacenamiento o red a RAM y lo inicia. Provee un Shell con comandos.
- ▶ **Linux Kernel:** se ejecuta desde RAM. Recupera el sistema completamente (los gestores de arranque ya no existen).



Arranque en ARM TI OMAP2+ / AM33xx



- ▶ **ROM Code:** intenta encontrar una imagen de arranque válida desde varias fuentes de almacenamiento y cargarlo en SRAM Tamaño limitado a <64 KB. No hay interacción del usuario posible.
- ▶ **X-Loader o U-Boot SPL:** se ejecuta desde SRAM. Inicializa la DRAM, el controlador NAND o MMC y carga el gestor de arranque secundario en la RAM y lo inicia. No es posible la interacción del usuario. Archivo llamado **MLO**.
- ▶ **U-Boot:** se ejecuta desde RAM. Inicializa otros dispositivos hardware (red, USB, etc.). Carga la imagen del núcleo desde almacenamiento o red a RAM y lo inicia. Provee un Shell de comandos. Archivo llamado **u-boot.bin** o **u-boot.img**.
- ▶ **Linux Kernel:** se ejecuta desde RAM. Toma completamente el control del sistema (el gestor de arranque no está más disponible).



- ▶ **ROM Code:** intenta encontrar una imagen de arranque válida desde varias fuentes de almacenamiento, y la carga en la RAM. La configuración de RAM se describe en un encabezado específico de la CPU, adjuntado a la imagen del gestor de arranque.
- ▶ **U-Boot:** se ejecuta desde RAM. Inicializa otros dispositivos hardware (red, USB, etc.). Carga la imagen del núcleo desde almacenamiento o red a RAM y lo inicia. Provee un Shell con comandos. Archivo llamado `u-boot.kwb`
- ▶ **Linux Kernel:** se ejecuta desde RAM. Recupera el sistema completamente (los gestores de arranque ya no existen)



Gestores de arranque genéricos

- ▶ Nos centraremos en la parte genérica, el cargador principal, que ofrece las características más importantes.
- ▶ Hay varios gestores de arranque genéricos de código abierto.
Aquí están los más populares:
 - ▶ **U-Boot**, el gestor de arranque universal de Denx
El más utilizado en ARM, también se utiliza en PPC, MIPS, x86, m68k, NIOS, etc.
El estándar de facto actual. Lo estudiaremos en detalle.
<http://www.denx.de/wiki/U-Boot>
 - ▶ **Barebox**, un nuevo gestor de arranque con arquitectura neutra, escrito como un sucesor de U-Boot. Mejor diseño, mejor código, desarrollo activo, pero aún no tiene tanto soporte de hardware como U-Boot.
<http://www.barebox.org>
- ▶ También hay muchas otras aplicaciones de código abierto de gestores de arranque, a menudo específicos de la arquitectura
 - ▶ RedBoot, Yaboot, PMON, etc.



El gestor de carga U-boot

U-Boot es un típico proyecto de software libre

- ▶ Licencia: GPLv2 (igual que Linux)
- ▶ Disponible de forma libre en <http://www.denx.de/wiki/U-Boot>
- ▶ Documentación disponible en
<http://www.denx.de/wiki/U-Boot/Documentation>
- ▶ El último código fuente de desarrollo está disponible en un repositorio Git:
<http://git.denx.de/?p=u-boot.git;a=summary>
- ▶ El desarrollo y las discusiones ocurren alrededor de una lista abierta
<http://lists.denx.de/pipermail/u-boot/>
- ▶ Desde finales de 2008, sigue un intervalo de versión fijo. Cada dos meses, se lanza una nueva versión. Las versiones se llaman YYYY.MM.



- ▶ Obtenga el código fuente del sitio web y descomprimalo
- ▶ El directorio `include/configs/` contiene un archivo de configuración para cada placa soportada
 - ▶ Define el tipo de CPU, los periféricos y su configuración, el mapeo de memoria, las características de U-Boot que se deben compilar, etc.
 - ▶ Se trata de un simple archivo `.h` que establece las constantes del preprocesador C. Vea el archivo `README` para la documentación de estas constantes. Este archivo también se puede ajustar para agregar o eliminar funciones de U-Boot (comandos, etc.).
- ▶ Suponiendo que su placa ya está soportada por U-Boot, debe haber una entrada correspondiente a su placa en el archivo `boards.cfg`.



Extracto del archivo de configuración de U-Boot

```
/* CPU configuration */
#define CONFIG_ARMV7 1
#define CONFIG_OMAP 1
#define CONFIG_OMAP34XX 1
#define CONFIG_OMAP3430 1
#define CONFIG_OMAP3_IGEP0020 1
[...]
/* Memory configuration */
#define CONFIG_NR_DRAM_BANKS 2
#define PHYS_SDRAM_1 OMAP34XX_SDRC_CS0
#define PHYS_SDRAM_1_SIZE (32 << 20)
#define PHYS_SDRAM_2 OMAP34XX_SDRC_CS1
[...]
/* USB configuration */
#define CONFIG_MUSB_UDC 1
#define CONFIG_USB_OMAP3 1
#define CONFIG_TWL4030_USB 1
[...]
/* Available commands and features */
#define CONFIG_CMD_CACHE
#define CONFIG_CMD_EXT2
#define CONFIG_CMD_FAT
#define CONFIG_CMD_I2C
#define CONFIG_CMD_MMC
#define CONFIG_CMD_NAND
#define CONFIG_CMD_NET
#define CONFIG_CMD_DHCP
#define CONFIG_CMD_PING
#define CONFIG_CMD_NFS
#define CONFIG_CMD_MTDPARTS
[...]
```



- ▶ U-Boot debe ser configurado antes de ser compilado
 - ▶ `make BOARDNAME_config`
 - ▶ Donde `BOARDNAME` es el nombre de la placa, como se ve en el archivo `boards.cfg` (primera columna).
- ▶ Asegúrese de que el compilador cruzado esté disponible en `PATH`
- ▶ Compile U-Boot, especificando el prefijo del compilador cruzado.
Por ejemplo, si el ejecutable del compilador cruzado es `arm-linux-gcc`:
`make CROSS_COMPILE=arm-linux-`
- ▶ El resultado principal es un archivo `u-boot.bin`, que es la imagen de U-Boot.
Dependiendo de su plataforma específica, puede haber otras imágenes especializadas: `u-boot.img`, `u-boot.kwb`, `MLO`, etc.



- ▶ Por lo general, U-Boot debe instalarse en la memoria flash para ser ejecutado por el hardware. Dependiendo del hardware, la instalación de U-Boot se realiza de una manera diferente:
 - ▶ La CPU proporciona algún tipo de monitor de arranque específico con el que puede comunicarse a través de un puerto serie o USB mediante un protocolo específico
 - ▶ La CPU arranca primero en medios extraíbles (MMC) antes de arrancar desde medios fijos (NAND). En este caso, arranca desde MMC para grabar una nueva versión
 - ▶ U-Boot ya está instalado y puede utilizarse para grabar una nueva versión de U-Boot. Sin embargo, tenga cuidado: si la nueva versión de U-Boot no funciona, la tarjeta es quedará inutilizable
 - ▶ La placa proporciona una interfaz JTAG, que permite escribir la memoria flash de forma remota, sin ningún sistema ejecutándose en la placa. También permite rescatar una placa si el gestor de arranque no funciona.



Mensaje de U-boot

- ▶ Conecte el destino al anfitrión a través de la consola serie
- ▶ Encienda la placa. En la consola serie va a ver algo como:

```
U-Boot 2013.04 (May 29 2013 - 10:30:21)
```

```
OMAP36XX/37XX-GP ES1.2, CPU-OPP2, L3-165MHz, Max CPU Clock 1 Ghz  
IGEPv2 + LPDDR/NAND
```

```
I2C: ready  
DRAM: 512 MiB  
NAND: 512 MiB  
MMC: OMAP SD/MMC: 0
```

```
Die ID #255000029ff800000168580212029011  
Net: smc911x-0  
U-Boot #
```

- ▶ El shell U-Boot ofrece un conjunto de comandos. Estudiaremos los más importantes, consulte la documentación para obtener una referencia completa o el comando `help`.



Comandos de información

Información Flash (NOR y SPI flash)

```
U-Boot> flinfo
DataFlash:AT45DB021
Nb pages: 1024
Page Size: 264
Size= 270336 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0001FFF (R0) Bootstrap
Area 1: C0002000 to C0003FFF Environment
Area 2: C0004000 to C0041FFF (R0) U-Boot
```

Información flash de NAND

```
U-Boot> nand info
Device 0: nand0, sector size 128 KiB
  Page size      2048 b
  OOB size       64 b
  Erase size    131072 b
```

Detalles de la versión

```
U-Boot> version
U-Boot 2013.04 (May 29 2013 - 10:30:21)
```



Comandos importantes 1/2

- ▶ El conjunto exacto de comandos depende de la configuración de U-Boot
- ▶ `help` and `help command`
- ▶ `boot`, ejecuta el comando por defecto de inicio, almacenado en `bootcmd`
- ▶ `bootm <address>`, inicia la imagen del kernel cargada en una dirección de memoria RAM dada
- ▶ `ext2load`, carga un archivo desde el sistema de archivos ext2 a la RAM
 - ▶ Y también `ext2ls` para ver los archivos, `ext2info` para información
- ▶ `fatload`, carga un archivo desde el sistema de archivos FAT a la RAM
 - ▶ También `fatls` y `fatinfo`
- ▶ `tftp`, carga un archivo desde la red a la RAM
- ▶ `ping`, para probar la red



Comandos importantes 2/2

- ▶ `loadb`, `loads`, `loady`, carga un archivo desde la linea serie en RAM
- ▶ `usb`, para inicializar y controlar el sistema USB, principalmente utilizado para dispositivos de almacenamiento USB como ser pendrives
- ▶ `mmc`, para inicializar y controlar el sistema MMC, utilizado por las tarjetas SD y microSD
- ▶ `nand`, para borrar, leer y escribir contenidos a la flash NAND
- ▶ `erase`, `protect`, `cp`, para borrar, proteger y escribir la flash NOR
- ▶ `md`, muestra los contenidos de la memoria. Puede ser de utilidad verificar los contenidos cargados en la memoria, o mirar los registros del Hardware.
- ▶ `mm`, modifica el contenido de la memoria. Puede ser útil para modificar directamente los registros de Hardware, para propósitos de prueba.



- ▶ U-Boot se puede configurar a través de variables de entorno, que afectan al comportamiento de los diferentes comandos.
- ▶ Las variables de entorno se cargan de flash a RAM al inicio de U-Boot, se puede modificar y guardar de nuevo en flash para persistencia
- ▶ Hay una ubicación dedicada en flash (o en el almacenamiento de MMC) para almacenar el entorno de U-Boot, definido en el archivo de configuración de la tarjeta



Comandos para manipular las variables de entorno:

- ▶ `printenv`
Muestra todas las variables
- ▶ `printenv <variable-name>`
Muestra el valor de una variable
- ▶ `setenv <variable-name> <variable-value>`
Cambia el valor de una variable, solo en RAM
- ▶ `editenv <variable-name>`
Modifica el valor de una variable, solo en RAM
- ▶ `saveenv`
Guarda el estado actual del entorno en flash



Comandos de variables de entorno - Ejemplo

```
u-boot # printenv  
baudrate=19200  
ethaddr=00:40:95:36:35:33  
netmask=255.255.255.0  
ipaddr=10.0.0.11  
serverip=10.0.0.1  
stdin=serial  
stdout=serial  
stderr=serial  
u-boot # printenv serverip  
serverip=10.0.0.1  
u-boot # setenv serverip 10.0.0.100  
u-boot # saveenv
```



Variables importantes del env de U-Boot

- ▶ `bootcmd`, contiene el comando que U-Boot ejecutará de forma automática al momento de iniciar después de una espera configurable, si el proceso no es interrumpido
- ▶ `bootargs`, contiene los argumentos pasados al kernel de Linux, contains the arguments passed to the Linux kernel, cubierto más tarde
- ▶ `serverip`, la dirección IP del servidor que U-Boot contactará para los comandos relacionados con la red
- ▶ `ipaddr`, la dirección IP que U-Boot va a usar
- ▶ `netmask`, la mascara de red para contactar al servidor
- ▶ `ethaddr`, la dirección MAC, se puede asignar solo una vez
- ▶ `bootdelay`, la espera en segundos antes de que U-Boot ejecute `bootcmd`
- ▶ `autostart`, si es si, U-Boot inicia automaticamente una imagen que haya sido cargada en memoria



- ▶ Las variables de entorno pueden contener pequeños scripts, para ejecutar varios comandos y probar los resultados de los comandos.
 - ▶ Util para automatizar el inicio o el proceso de mejora
 - ▶ Varios comandos pueden ser enganchados utilizando el operador ;
 - ▶ Las pruebas se pueden realizar haciendo

```
if command ; then ... ; else ... ; fi
```
 - ▶ Los Scripts se ejecutan haciendo run <variable-name>
 - ▶ Se pueden referenciar otras variables haciendo \${variable-name}
- ▶ Ejemplo
 - ▶

```
setenv mmc-boot 'if fatload mmc 0 80000000 boot.ini; then source;
else if fatload mmc 0 80000000 uImage; then run mmc-
bootargs; bootm; fi; fi'
```



- ▶ U-Boot se usa principalmente para cargar e iniciar una imagen del kernel, pero también permite cambiar la imagen del kernel y el sistema de archivos raíz almacenado en flash
- ▶ Los archivos deben intercambiarse entre el destino y la estación de trabajo de desarrollo. Esto es posible:
 - ▶ A través de la red si el destino tiene una conexión Ethernet y U-Boot contiene un controlador para el chip Ethernet. Esta es la solución más rápida y eficiente.
 - ▶ A través de una llave USB, si U-Boot soporta el controlador USB de su plataforma
 - ▶ A través de una tarjeta SD o microSD, si U-Boot admite el controlador MMC de su plataforma
 - ▶ A través del puerto serie



- ▶ La transferencia sobre la red desde la estación de desarrollo y U-Boot en el destino se realiza a través de TFTP
 - ▶ *Trivial File Transfer Protocol*
 - ▶ Algo similar a FTP, pero sin autenticación sobre UDP
- ▶ Un servidor TFTP es necesario en la estación de trabajo de desarrollo
 - ▶ `sudo apt-get install tftpd-hpa`
 - ▶ Todos los archivos en `/var/lib/tftpboot` son visibles a través de TFTP
 - ▶ Un cliente TFTP está disponible en el paquete `tftp-hpa`, para pruebas
- ▶ Un cliente TFTP se encuentra integrado con U-Boot
 - ▶ Configure las variables de entorno `ipaddr` y `serverip`
 - ▶ Utilice `tftp <address> <filename>` para cargar un archivo



mkimage de U-boot

- ▶ La imagen del núcleo que U-Boot carga e inicia debe estar preparado, de modo que se añade un encabezado específico de U-Boot delante de la imagen
 - ▶ Este encabezado proporciona detalles como el tamaño de la imagen, la dirección de carga esperada, el tipo de compresión, etc.
- ▶ Esto se hace con una herramienta que viene en U-Boot, `mkimage`
- ▶ Debian / Ubuntu: simplemente instale el paquete `u-boot-tools`.
- ▶ O, compílelo usted mismo: simplemente configure U-Boot para cualquier placa de cualquier arquitectura y compile. A continuación, instale `mkimage`:
`cp tools/mkimage /usr/local/bin/`
- ▶ El destino especial `uImage` del Makefile del kernel puede luego ser utilizado para generar una imagen de kernel adecuada para U-Boot.



Es momento de iniciar la práctica de laboratorio!

- ▶ Conectarse con la placa utilizando la consola serie
- ▶ Configurar, construir e instalar *U-Boot*
- ▶ Aprender los comandos *U-Boot*
- ▶ Preparar la comunicación *TFTP* con la placa



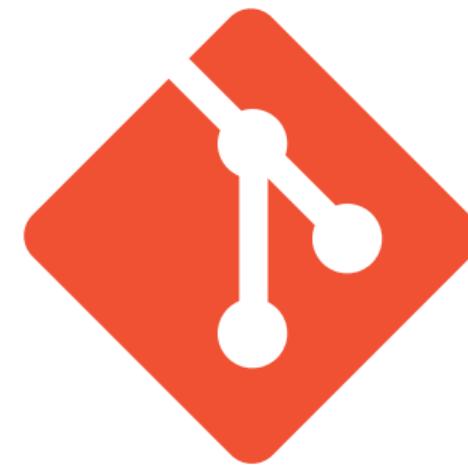
Introducción al kernel de Linux

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Características de Linux



- ▶ El kernel de Linux es un componente del sistema, que también requiere de bibliotecas y aplicaciones para proporcionar características a usuarios finales.
- ▶ El kernel de Linux fue creado como un hobby en 1991 por el estudiante Finlandés, Linus Torvalds.
 - ▶ Linux comienza a ser utilizado rápidamente como kernel en Software de sistemas operativos libres.
- ▶ Linus Torvalds ha sido capaz de crear una comunidad grande y dinámica de desarrolladores y usuarios en Linux.
- ▶ Hoy en día, más de mil personas contribuyen a cada lanzamiento del kernel, entre individuos, empresas grandes y pequeñas



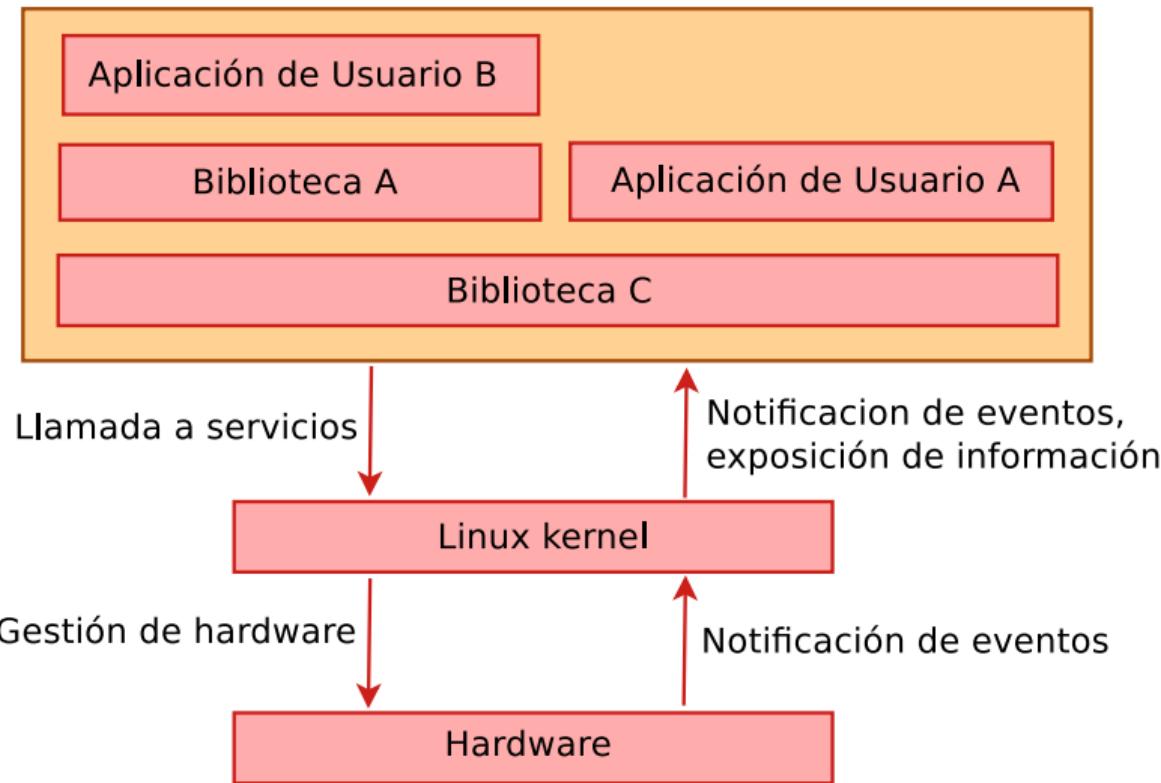
Linus Torvalds in 2014
Image credits (Wikipedia):
<https://bit.ly/2UIa1TD>



- ▶ Portabilidad y soporte de hardware. Se ejecuta en la mayoría de las arquitecturas.
- ▶ Escalabilidad. Se puede ejecutar en super computadoras, así como en dispositivos mínimos (4 MB de RAM es suficiente).
- ▶ Cumplimiento de estándares e interoperabilidad.
- ▶ Exhaustivo soporte de red.
- ▶ Seguridad. No puede ocultar sus defectos. Su código es revisado por muchos expertos.
- ▶ Estabilidad y confiabilidad.
- ▶ Modularidad. Puede incluir solo lo que el sistema necesita incluso en tiempo de ejecución.
- ▶ Fácil de programar. Puedes aprender del código existente. Muchos recursos útiles en la red.



El Kernel de Linux en el sistema





- ▶ **Administra todos los recursos de hardware:** CPU, memoria, E/S.
- ▶ Provee un **juego portable, de APIs independientes de arquitectura y Hardware** para permitir a las aplicaciones del espacio de usuario y bibliotecas utilizar recursos de Hardware.
- ▶ **Maneja el acceso y uso concurrente** de recursos de Hardware desde diferentes aplicaciones.
 - ▶ Ejemplo: una interfaz de red simple se utiliza por multiples aplicaciones en espacio de usuario a través de varias conexiones de red. El kernel es responsable de “multiplexar” los recursos de Hardware.



Llamadas al sistema

- ▶ La interface principal entre el kernel y el espacio de usuario es un juego de llamadas al sistema
- ▶ Al rededor de 400 llamadas al sistema proveen los servicios principales del kernel
 - ▶ Operaciones sobre archivos y dispositivos, operaciones de red, comunicación entre procesos, gestión de procesos, mapeo de memoria, temporizadores, hilos, sincronización de primitivas, etc.
- ▶ Esta interfaz es estable a través del tiempo: solo nuevas llamadas al sistema se pueden agregar por los desarrolladores del kernel
- ▶ Esta interfaz de llamadas al sistema esta enmascarada por la biblioteca C, y las aplicaciones del espacio de usuario generalmente nunca hacen una llamada al sistema directa, pero en su lugar usan su correspondiente función de la biblioteca C



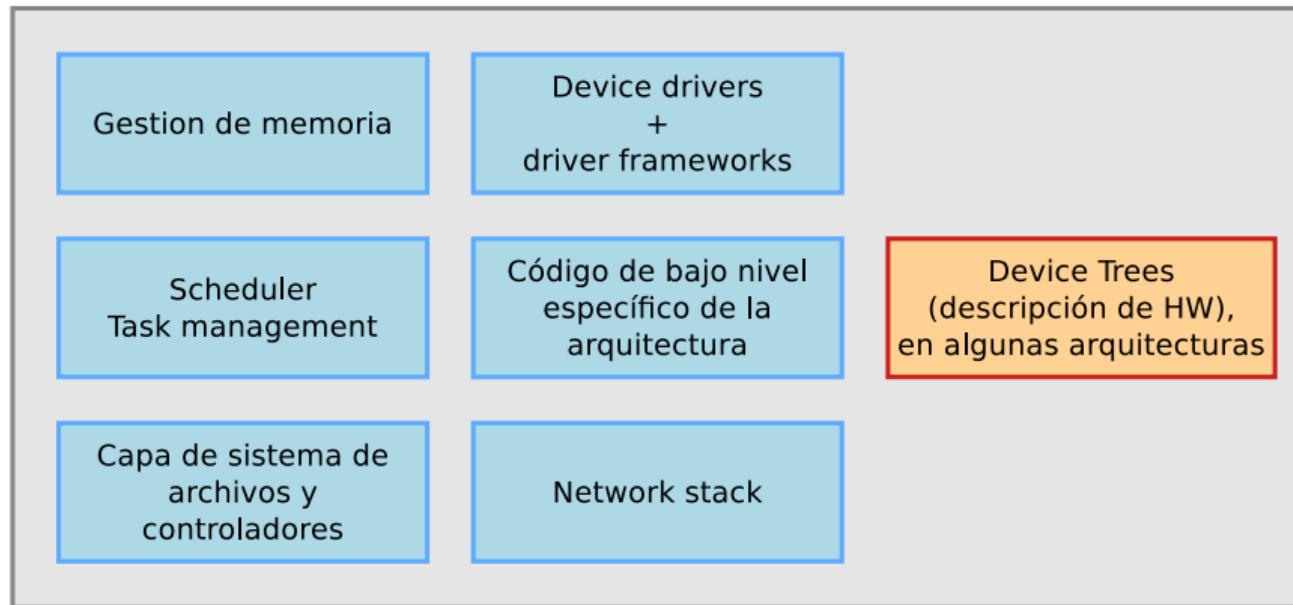
Image credits
(Wikipedia):
<https://bit.ly/2U2rdGB>



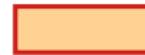
- ▶ Linux provee de informacion sobre el sistema y el Kernel en el espacio de usuario a traves de los **pseudo sistemas de archivos**, algunas veces llamado **sistema de archivo virtual**
- ▶ Los Pseudo sistema de archivos permiten a las aplicaciones ver directorios y archivos que no existen en realidad en ningun almacenamiento real: son creados y actualizados en el momento por el kernel
- ▶ Los dos pseudo sistemas de archivos mas importantes son
 - ▶ `proc`, generalmente montado en `/proc`:
La informacion relativa al sistema operativo (procesos, parametros de manejo de memoria...)
 - ▶ `sysfs`, generalmente montado en `/sys`:
Representa al sistema como un juego de dispositivos y buses. Informacion acerca de los dispositivos.



Kernel de Linux



Implementado principalmente en C, con algo de assembly.



Escrito en un lenguaje específico del Device Tree.



- ▶ Todas las fuentes de Linux son Software Libre lanzado bajo la Licencia Pública General GNU versión 2 (GPL v2).
- ▶ Para el kernel de Linux, esto básicamente implica que:
 - ▶ Cuando recibe o compra un dispositivo con Linux, debe recibir las fuentes de Linux, con el derecho de estudiarlas, modificarlas y redistribuirlas.
 - ▶ Cuando produce dispositivos basados en Linux, debe liberar los fuentes al destinatario, con los mismos derechos, sin restricción.



- ▶ Vea el directorio `arch` en los fuentes del kernel
- ▶ Mínimo: procesadores de 32 bit, con o sin MMU, y soporte para `gcc`
- ▶ arquitecturas 32 bit (subdirectorios de `arch`)
Ejemplos: `arm`, `avr32`, `blackfin`, `c6x`, `m68k`, `microblaze`, `mips`, `score`, `sparc`, `um`
- ▶ arquitecturas 64 bit:
Ejemplos: `alpha`, `arm64`, `ia64`, `tile`
- ▶ arquitecturas 32/64 bit
Ejemplos: `powerpc`, `x86`, `sh`, `sparc`
- ▶ Busque detalles en los fuentes del kernel: `arch/<arch>/Kconfig`,
`arch/<arch>/README`, o `Documentation/<arch>/`



Esquema de versiones de Linux y proceso de desarrollo

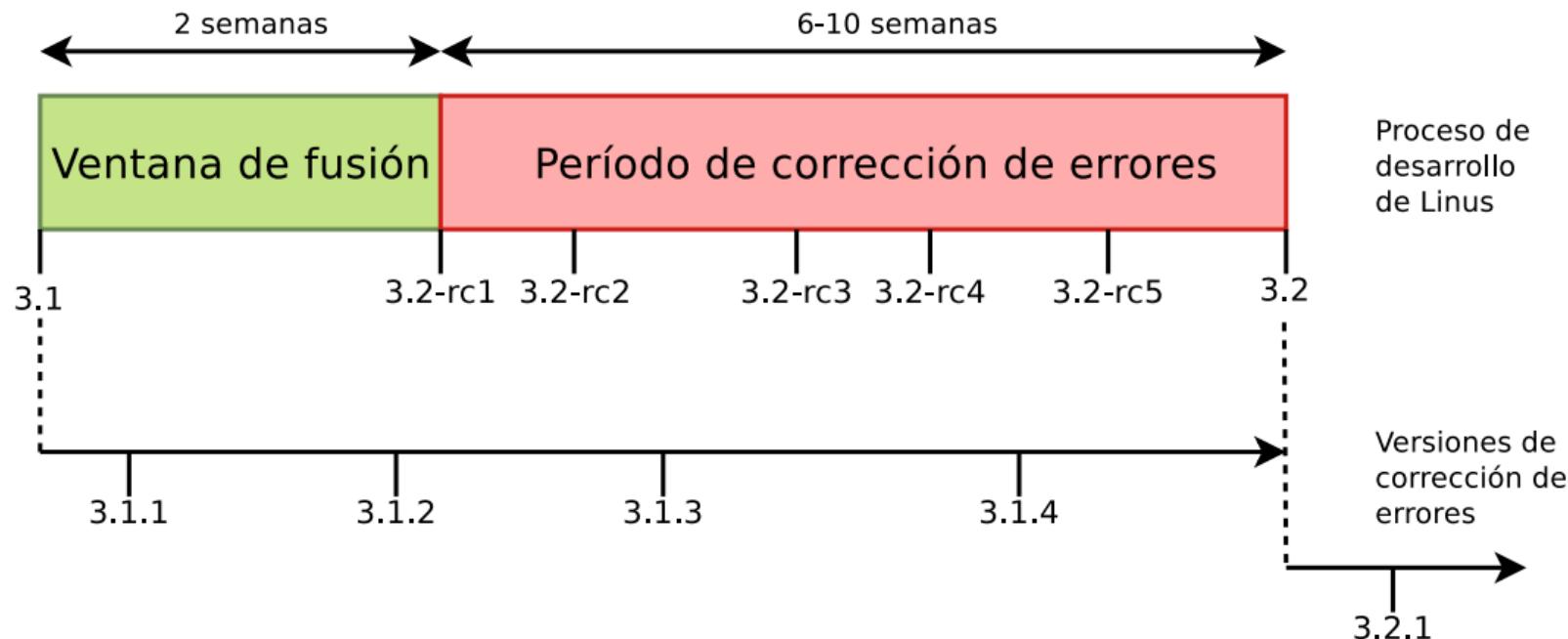


- ▶ Hasta 2003, había una nueva rama de lanzamiento estable de Linux cada 2 o 3 años (2.0, 2.2, 2.4). Las nuevas ramas de desarrollo tardaron 2-3 años en estabilizarse (¡demasiado lento!).
- ▶ Desde 2003, hay una nueva versión estable de Linux cada 10 semanas:
 - ▶ Versión 2.6 (Dic. 2003) a 2.6.39 (May 2011)
 - ▶ Versión 3.0 (Jul. 2011) a 3.19 (Feb. 2015)
 - ▶ Versión 4.0 (Abr. 2015) a 4.20 (Dic. 2018)
 - ▶ Versión 5.0 fue lanzado en marzo de 2019.
- ▶ Las características se agregan al kernel de manera progresiva. Desde 2003, los desarrolladores de kernel han logrado hacerlo sin tener que introducir una rama de desarrollo masivamente incompatible.
- ▶ Para cada versión, hay correcciones de errores y actualizaciones de seguridad: 5.0.1, 5.0.2, etc.



Nuevo modelo de desarrollo

Uso de ventanas de fusión y corrección de errores





Necesidad de soporte a largo plazo

- ▶ Problema: las correcciones de errores y seguridad solo se lanzaron para las versiones más recientes de kernel estable. Solo las versiones LTS (*Long Term Support*) son compatibles por hasta 6 años.
- ▶ Ejemplo en Google: a partir de *Android O*, todos los dispositivos Android nuevos deberán ejecutar dicho núcleo LTS.
- ▶ También puede obtener soporte a largo plazo de un proveedor comercial Linux incorporado
- ▶ El proyecto *Civil Infrastructure Platform* es un esfuerzo de la industria / Linux Foundation para admitir versiones LTS seleccionadas (comenzando con 4.4) mucho más tiempo (> 10 años). Ver <http://bit.ly/2hy1QYC>.

Longterm release kernels

Version	Maintainer	Released	Projected EOL
4.19	Greg Kroah-Hartman	2018-10-22	Dec, 2020
4.14	Greg Kroah-Hartman	2017-11-12	Jan, 2020
4.9	Greg Kroah-Hartman	2016-12-11	Jan, 2023
4.4	Greg Kroah-Hartman	2016-01-10	Feb, 2022
3.16	Ben Hutchings	2014-08-03	Apr, 2020

Fuente: siga el enlace "Releases" en <https://kernel.org> (14 de Marzo de 2019)



¿Qué hay de nuevo en cada versión de Linux? (1)

¡La lista oficial de cambios para cada versión de Linux es una enorme lista de parches individuales!

```
commit aa6e52a35d388e730f4df0ec2ec48294590cc459
Author: Thomas Petazzoni <thomas.petazzoni@bootlin.com>
Date:   Wed Jul 13 11:29:17 2011 +0200

at91: at91-ohci: support overcurrent notification

Several USB power switches (AIC1526 or MIC2026) have a digital output
that is used to notify that an overcurrent situation is taking
place. This digital outputs are typically connected to GPIO inputs of
the processor and can be used to be notified of these overcurrent
situations.

Therefore, we add a new overcurrent_pin[] array in the at91_usbhs_data
structure so that boards can tell the AT91 OHCI driver which pins are
used for the overcurrent notification, and an overcurrent_supported
boolean to tell the driver whether overcurrent is supported or not.

The code has been largely borrowed from ohci-da8xx.c and
ohci-s3c2410.c.

Signed-off-by: Thomas Petazzoni <thomas.petazzoni@bootlin.com>
Signed-off-by: Nicolas Ferre <nicolas.ferre@atmel.com>
```

Es muy difícil encontrar los cambios claves y obtener una imagen global de los cambios individuales.



¿Qué hay de nuevo en cada versión de Linux? (2)

Afortunadamente, hay algunos recursos útiles disponibles

- ▶ <https://kernelnewbies.org/LinuxChanges>
Cobertura profunda de las nuevas características en cada versión del kernel
- ▶ <http://lwn.net>
Cobertura de las funciones aceptadas en cada ventana de fusión
- ▶ <http://www.linux-arm.info>
Noticias sobre Linux en ARM, incluidos los cambios en el kernel.
- ▶ <http://linuxfr.org>, for French readers. También hay un largo resumen de nuevas características (diferentes del contenido en LinuxChanges).



Fuentes del kernel de Linux



Ubicación de las fuentes del núcleo

- ▶ Las versiones oficiales *mainline* del kernel de Linux, según lo publicado por Linus Torvalds, están disponibles en <https://kernel.org>
 - ▶ Estas versiones siguen el modelo de desarrollo del núcleo
 - ▶ Sin embargo, es posible que todavía no contengan el último desarrollo de un área específica. Es posible que algunas características del desarrollo aún no estén listas para la inclusión en la línea principal
- ▶ Muchos proveedores de chips suministran sus propias fuentes de kernel
 - ▶ Centrándose primero en el soporte de hardware
 - ▶ Puede tener un delta muy importante con Linux principal
 - ▶ Útil solo cuando la línea principal aún no se ha puesto al día
- ▶ Muchas subcomunidades del núcleo mantienen su propio núcleo, con características generalmente más nuevas pero menos estables
 - ▶ Comunidades de arquitectura (ARM, MIPS, PowerPC, etc.), comunidades de controladores de dispositivos (I2C, SPI, USB, PCI, red, etc.), otras comunidades (en tiempo real, etc.)
 - ▶ No hay lanzamientos oficiales, solo hay árboles de desarrollo disponibles.



- ▶ Las fuentes del núcleo están disponibles en
<https://kernel.org/pub/linux/kernel> como **full tarballs** (fuentes completas del núcleo) y **parches** (diferencias entre dos versiones del núcleo).
- ▶ Sin embargo, cada vez más personas usan el sistema de control de versiones **git**. ¡Absolutamente necesario para el desarrollo del kernel!
 - ▶ Obtener todas las fuentes e historial del kernel

```
git clone git:  
//git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```
 - ▶ Cree una rama que comience en una versión estable específica

```
git checkout -b <name-of-branch> v3.11
```
 - ▶ Interfaz web disponible en <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/>.
 - ▶ Lea más sobre Git en <https://git-scm.com/>



Trabajando con git: se necesita almacenamiento SSD

Para todo el trabajo con `git`, pero especialmente en grandes proyectos como el kernel de Linux ...

- ▶ Tener un disco rápido acelerará dramáticamente la mayoría de las operaciones `git`.
- ▶ Pídale a su jefe que solicite un disco SSD para su computadora portátil. Te hará más productivo.





Tamaño del kernel de Linux (1)

- ▶ Fuentes de Linux 4.11:
 - ▶ 57994 archivos (`git ls-files | wc -l`)
 - ▶ 23144003 líneas (`wc -l $(git ls-files)`)
 - ▶ 675576310 bytes (`wc -c $(git ls-files)`)
- ▶ Tamaño mínimo de kernel compilado de Linux 4.11, arranque en la placa versátil ARM (disco duro en PCI, sistema de archivos ext2, soporte ejecutable ELF, consola framebuffer y dispositivos de entrada):
876 KB (comprimido), 2.3 MB (sin formato)
- ▶ ¿Por qué son tan grandes los fuentes?
Debido a que incluyen miles de controladores de dispositivos, muchos protocolos de red, admiten muchas arquitecturas y sistemas de archivos ...
- ▶ ¡El núcleo de Linux (planificador, gestión de memoria ...) es bastante pequeño!



Tamaño del kernel de Linux (2)

A partir de la versión del kernel 4.6 (en porcentaje de número de líneas).

- ▶ drivers: 57.0%
- ▶ arch: 16.3%
- ▶ fs: 5.5%
- ▶ sound: 4.4%
- ▶ net: 4.3%
- ▶ include: 3.5%
- ▶ Documentation: 2.8%
- ▶ tools: 1.3%
- ▶ kernel: 1.2%
- ▶ firmware: 0.6%
- ▶ lib: 0.5%
- ▶ mm: 0.5%
- ▶ scripts: 0.4%
- ▶ crypto: 0.4%
- ▶ security: 0.3%
- ▶ block: 0.1%
- ▶ ...



▶ Tarballs completos

- ▶ Contiene las fuentes completas del kernel: largas para descargar y descomprimir, pero debe hacerse al menos una vez

- ▶ Ejemplo:

<https://kernel.org/pub/linux/kernel/v4.x/linux-4.20.13.tar.xz>

- ▶ Comandos de extracción:

```
tar xf linux-4.20.13.tar.xz
```

▶ Parches incrementales entre versiones

- ▶ Se supone que ya tiene una versión base y aplica los parches correctos en el orden correcto. Rápido para descargar y aplicar

- ▶ Ejemplos:

[\(4.19 to 4.20\)](https://kernel.org/pub/linux/kernel/v4.x/patch-4.20.xz)

[\(4.20 to 4.20.13\)](https://kernel.org/pub/linux/kernel/v4.x/patch-4.20.13.xz)

- ▶ Todas las versiones anteriores del kernel están disponibles en
<https://kernel.org/pub/linux/kernel/>



- ▶ Un parche es la diferencia entre dos árboles fuente
 - ▶ Calculado con la herramienta code diff, o con sistemas de control de versiones más elaborados
- ▶ Son muy comunes en la comunidad de código abierto.
- ▶ Extracto de un parche:

```
diff -Nru a/Makefile b/Makefile
--- a/Makefile 2005-03-04 09:27:15 -08:00
+++ b/Makefile 2005-03-04 09:27:15 -08:00
@@ -1,7 +1,7 @@
 VERSION = 2
 PATCHLEVEL = 6
 SUBLEVEL = 11
-EXTRAVERSION =
+EXTRAVERSION = .1
 NAME=Woozy Numbat

# *DOCUMENTATION*
```



Contenido de un parche

- ▶ Una sección por archivo modificado, comenzando con un encabezado

```
diff -Nru a/Makefile b/Makefile
--- a/Makefile 2005-03-04 09:27:15 -08:00
+++ b/Makefile 2005-03-04 09:27:15 -08:00
```

- ▶ Una subsección por parte modificada del archivo, comenzando con el encabezado con los números de línea afectados

```
@@ -1,7 +1,7 @@
```

- ▶ Tres líneas de contexto antes del cambio

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 11
```

- ▶ El cambio en sí

```
-EXTRAVERSION =
+EXTRAVERSION = .1
```

- ▶ Tres líneas de contexto después del cambio

```
NAME=Woozy Numbat
```

```
# *DOCUMENTATION*
```



Usando el comando parche

El comando patch:

- ▶ Toma el contenido del parche en su entrada estándar
- ▶ Aplica las modificaciones descritas por el parche en el directorio actual

patch ejemplos de uso:

- ▶ `patch -p<n> < diff_file`
- ▶ `cat diff_file | patch -p<n>`
- ▶ `xzcat diff_file.xz | patch -p<n>`
- ▶ `zcat diff_file.gz | patch -p<n>`
- ▶ Notas:
 - ▶ n: cantidad de niveles de directorio para omitir en las rutas de archivo
 - ▶ Puede revertir aplicar un parche con la opción `-R`
 - ▶ Puede probar un parche con la opción `--dry-run`



Aplicando un parche de Linux

- ▶ Dos tipos de parches de Linux:
 - ▶ Ya sea para ser aplicado a la versión estable anterior
(de `x.<y-1>` a `x.y`)
 - ▶ O implementar correcciones a la versión estable actual
(de `x.y` a `x.y.z`)
- ▶ Se puede descargar en `gzip`, `bzip2` o `xz` (archivos mucho más pequeños) comprimidos.
- ▶ Siempre producido para `n=1`
(eso es lo que todos hacen... ¡Hazlo también!)
- ▶ Necesita ejecutar el comando `patch` dentro del **directorio de origen** del kernel
- ▶ Ejemplo de línea de comando de parche de Linux:

```
cd linux-4.19
xzcat ../patch-4.20.xz | patch -p1
xzcat ../patch-4.20.13.xz | patch -p1
cd ..; mv linux-4.19 linux-4.20.13
```



¡Es hora de comenzar el laboratorio práctico!

- ▶ Obtenga las fuentes del kernel de Linux
- ▶ Aplicar parches



Configuración del Kernel



- ▶ La configuración del kernel y el sistema de construcción se basa en multiples archivos Makefile
- ▶ Uno interactua solo con el Makefile principal, presente en el **directorio superior** de los fuentes del kernel
- ▶ La interacción tiene lugar
 - ▶ utilizando la herramienta `make`, que analiza el Makefile
 - ▶ mediante varios **objetivos**, definiendo que acción deberá hacerse (configuración, compilación, etc.). Ejecutar `make help` para ver todos los objetivos disponibles.
- ▶ Ejemplo
 - ▶ `cd linux-3.6.x/`
 - ▶ `make <target>`



Configuración del Kernel (1)

- ▶ El núcleo contiene miles de controladores de dispositivo, controladores de sistema de archivos, protocolos de red y otros elementos configurables.
- ▶ Hay miles de opciones disponibles, que se utilizan para compilar selectivamente partes del código fuente del núcleo
- ▶ La configuración del kernel es el proceso de definir el conjunto de opciones con las que desea que se compile su kernel
- ▶ El conjunto de opciones depende
 - ▶ En su hardware (para controladores de dispositivo, etc.)
 - ▶ Sobre las capacidades que le gustaría dar a su núcleo (capacidades de red, sistemas de archivos, en tiempo real, etc.)



Configuración del Kernel (2)

- ▶ La configuración se almacena en el archivo `.config` en la raíz de las fuentes del núcleo
 - ▶ Archivo de texto simple, estilo `key=value`
- ▶ Como las opciones tienen dependencias, generalmente nunca se editan a mano, sino a través de interfaces gráficas o de texto:
 - ▶ `make xconfig`, `make gconfig` (gráfica)
 - ▶ `make menuconfig`, `make nconfig` (texto)
 - ▶ Puede cambiar de uno a otro, todos cargan/guardan el mismo archivo `.config` y muestran el mismo conjunto de opciones
- ▶ Para modificar un kernel en una distribución GNU/Linux: los archivos de configuración generalmente se publican en `/boot/`, junto con las imágenes del kernel: `/boot/config-3.2.0-31-generic`



¿Kernel o módulo?

- ▶ La **imagen del kernel** es un **archivo único**, resultante de la vinculación de todos los archivos de objetos que corresponden a las características habilitadas en la configuración
 - ▶ Este es el archivo que el gestor de arranque carga en la memoria
 - ▶ Por lo tanto, todas las funciones incluidas están disponibles tan pronto como se inicia el núcleo, en un momento en que no existe un sistema de archivos
- ▶ Sin embargo, algunas características (controladores de dispositivos, sistemas de archivos, etc.) pueden compilarse como **módulos**
 - ▶ Estos son *plugins* que se pueden cargar/descargar dinámicamente para agregar/eliminar funciones al núcleo
 - ▶ Cada **módulo se almacena como un archivo separado en el sistema de archivos** y, por lo tanto, el acceso a un sistema de archivos es obligatorio para usar módulos
 - ▶ Esto no es posible en el procedimiento de arranque temprano del núcleo, porque no hay un sistema de archivos disponible



Tipos de opciones de kernel

- ▶ Hay diferentes tipos de opciones
 - ▶ opciones `bool`, son
 - ▶ *verdaderas* (para incluir la función en el núcleo) o
 - ▶ *falsas* (para excluir la función del núcleo)
 - ▶ opciones `tristate`, son
 - ▶ *verdaderas* (para incluir la función en la imagen del núcleo) o
 - ▶ *módulo* (para incluir la función como un módulo del núcleo) o
 - ▶ *falsas* (para excluir la función)
 - ▶ opciones `int`, para especificar valores enteros
 - ▶ opciones `hex`, para especificar valores hexadecimales
 - ▶ opciones `string`, para especificar valores de cadena



- ▶ Hay dependencias entre las opciones del kernel
- ▶ Por ejemplo, habilitar un controlador de red requiere que la pila de red esté habilitada
- ▶ Dos tipos de dependencias
 - ▶ `depends` dependencias. En este caso, la opción A que depende de la opción B no es visible hasta que se habilita la opción B
 - ▶ `select` dependencias. En este caso, con la opción A dependiendo de la opción B, cuando la opción A está habilitada, la opción B se habilita automáticamente
 - ▶ `make xconfig` permite ver todas las opciones, incluso las que no se pueden seleccionar debido a la falta de dependencias. En este caso, se muestran en gris



make xconfig

- ▶ La interfaz gráfica más común para configurar el núcleo
- ▶ Asegúrate de leer

```
help -> introduction: useful options!
```
- ▶ Explorador de archivos: más fácil de cargar archivos de configuración
- ▶ Interfaz de búsqueda para buscar parámetros
- ▶ Paquetes necesarios de Debian/Ubuntu: libqt4-dev g++ (libqt3-mt-dev para versiones anteriores del kernel)



Capturas de pantalla de make xconfig

The screenshot shows the 'Linux/arm 3.4.0 Kernel Configuration' window. The left pane displays a tree view of kernel configuration options, while the right pane provides detailed information about selected options.

Left Pane (Option Menu):

- General setup
 - IRQ subsystem
 - RCU Subsystem
 - Control Group support
 - Namespaces support
 - Configure standard kernel features (expert users)
 - Kernel Performance Events And Counters
 - GCOV-based kernel profiling
 - Enable loadable module support
- Enable the block layer
 - Partition Types
 - IO Schedulers
- System Type
 - TI OMAP2/3/4 Specific Features
- Bus support
 - OPCCard (PCMCIA/CardBus) support
- Kernel Features
- Boot options
- CPU Power Management
 - CPU Frequency scaling
 - Floating point emulation
 - Userspace binary formats
 - Power management options
- Networking support
 - Networking options

Right Pane (Detailed View):

Selected Option: TI OMAP2/3/4 (ARCH_OMAP2PLUS)

Symbol: CONFIG_ARCH_OMAP2PLUS

Type: boolean

Prompt: TI OMAP2/3/4

Defined at: arch/arm/plat-omap/Kconfig:24

Depends on: <choice>

Location:

- > System Type
- > TI OMAP Common Features
- > OMAP System Type (<choice> [=y])



Interfaz de búsqueda de make xconfig

Busca una palabra clave en el nombre del parámetro. Permite seleccionar o deseleccionar los parámetros encontrados.

Search Config

Find: mtd

Option

- (0) Physical address of DiskOnChip
 - NAND Flash support for Samsung S3C SoCs
 - Support software BCH ECC
 - ST Nomadik 8815 NAND support
 - CFI Flash device mapped on AMD NetSc520
 - M-Systems Disk-On-Chip Millennium-only alternative driver (DEPRECATED)
 - ARM Firmware Suite partition parsing (NEW)
 - PMC551 Debugging
 - Command line partition table parsing

Physical address of DiskOnChip (MTD_DOCPROBE_ADDRESS)

CONFIG_MTD_DOCPROBE_ADDRESS:

By default, the probe for DiskOnChip devices will look for a DiskOnChip at every multiple of 0x2000 between 0xC8000 and 0xEE000. This option allows you to specify a single address at which to probe for the device, which is useful if you have other devices in that range which get upset when they are probed.



Opciones de configuración del Kernel

Compiled as a module (separate file)

`CONFIG_IS09660_FS=m`

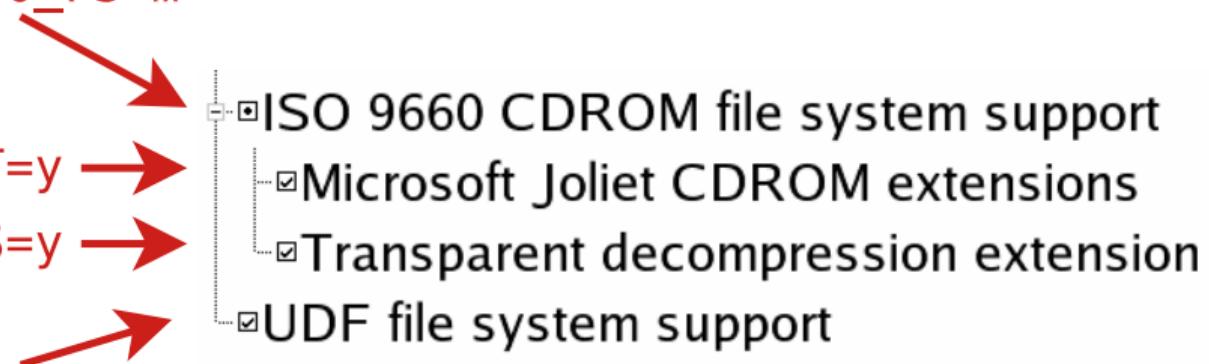
Driver options

`CONFIG_JOLIET=y`

`CONFIG_ZISOFS=y`

Compiled statically into the kernel

`CONFIG_UDF_FS=y`





Extracto de archivo .config correspondiente

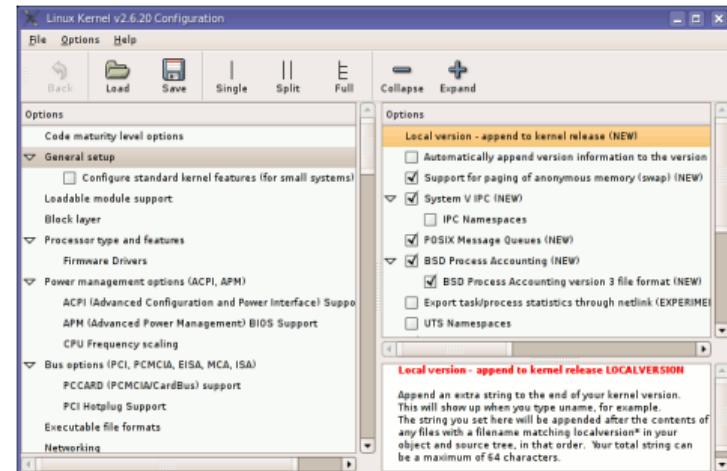
Las opciones están agrupadas por secciones y tienen el prefijo CONFIG_.

```
#  
# 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
```



make gconfig

- ▶ Interfaz de configuración gráfica basada en *GTK*. Funcionalidad similar a la de make *xconfig*.
- ▶ Solo falta una funcionalidad de búsqueda.
- ▶ Paquetes necesarios de Debian:
libglade2-dev

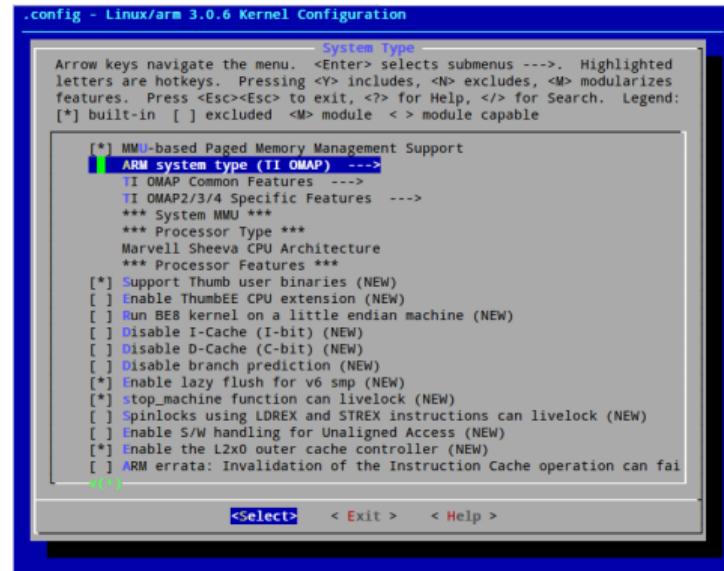




make menuconfig

make menuconfig

- ▶ Útil cuando no hay gráficos disponibles. ¡Muy conveniente también!
- ▶ La misma interfaz que se encuentra en otras herramientas: BusyBox, Buildroot ...
- ▶ Paquetes necesarios de Debian:
`libncurses-dev`



make nconfig

- ▶ Una nueva interfaz de texto similar
- ▶ Más fácil de usar (por ejemplo, más fácil acceder a la información de ayuda).
- ▶ Paquetes necesarios de Debian:
libncurses-dev

```
...config - Linux/x86_64 3.0.0 Kernel Configuration
Linux/x86_64 3.0.0 Kernel Configuration

[ ] General setup --->
  [ ] Enable loadable module support --->
    *. Enable the block layer --->
      Processor type and features --->
      Power management and ACPI options --->
      Bus options (PCI etc.) --->
      Executable file formats / Emulations --->
  [ ] Networking support --->
    Device Drivers --->
    Firmware Drivers --->
    File systems --->
    Kernel hacking --->
    Security options --->
  [ ] Cryptographic API --->
  [ ] Virtualization --->
    Library routines --->

F1 Help - F2 Sym Info - F3 Insts - F4 Config - F5 Back - F6 Save - F7 Load - F8 Sym Search - F9 Exit
```



make oldconfig

- ▶ Necesario muy a menudo!
- ▶ Útil para actualizar un archivo `.config` de una versión anterior del kernel
- ▶ Emite advertencias para los parámetros de configuración que ya no existen en el nuevo núcleo.
- ▶ Pide valores para nuevos parámetros

Si edita un archivo `.config` a mano, se recomienda encarecidamente ejecutar `make oldconfig` después.



Un problema frecuente:

- ▶ Después de cambiar varias configuraciones de kernel, su kernel ya no funciona
- ▶ Si no recuerda todos los cambios que realizó, puede volver a su configuración anterior:
`$ cp .config.old .config`
- ▶ Todas las interfaces de configuración del núcleo (`xconfig`, `menuconfig`, `oldconfig` ...) mantienen esta copia de seguridad de `.config.old`.



- ▶ El conjunto de opciones de configuración depende de la arquitectura
 - ▶ Algunas opciones de configuración son muy específicas de la arquitectura
 - ▶ La mayoría de las opciones de configuración (opciones de núcleo global, subsistema de red, sistemas de archivos, la mayoría de los controladores de dispositivo) son visibles en todas las arquitecturas.
- ▶ De manera predeterminada, el sistema de compilación del núcleo supone que el núcleo se está construyendo para la arquitectura del host, es decir, la compilación nativa
- ▶ La arquitectura no está definida dentro de la configuración, sino a un nivel superior
- ▶ Más adelante veremos cómo anular este comportamiento, para permitir la configuración de núcleos para una arquitectura diferente



Compilando e instalando el Kernel para el sistema anfitrion



- ▶ make
 - ▶ en el directorio fuente principal de núcleo
 - ▶ Recuerde ejecutar multiples trabajos en paralelo si es que posee multiples núcleos.
Ejemplo: `make -j 4`
 - ▶ No es necesario ser administrador!
- ▶ Esto genera
 - ▶ `vmlinu`x, la imagen del kernel en bruto sin comprimir, en formato ELF, útil para fines de depuración, pero no puede ser iniciado
 - ▶ `arch/<arch>/boot/*Image`, la imagen final, por lo general comprimida del kernel que puede ser iniciada
 - ▶ `bzImage` para x86, `zImage` para ARM, `vmImage.gz` para Blackfin, etc.
 - ▶ `arch/<arch>/boot/dts/*.dtb`, archivos compilados del Device Tree (en algunas arquitecturas)
 - ▶ Todos los módulos del kernel, extendido sobre el ábol de fuentes del kernel, como archivos `.ko`.



- ▶ make install
 - ▶ Realiza por defecto la instalación para el sistema anfitrón, por lo tanto necesita ser ejecutado como root. Generalmente no se utiliza cuando se compila para sistemas embebidos, ya que instala los archivos en la estación de desarrollo.
- ▶ Instala
 - ▶ /boot/vmlinuz-<version>
Imagen del kernel comprimida. Al igual que el que esta en arch/<arch>/boot
 - ▶ /boot/System.map-<version>
Almacena las direcciones de simbolos del kernel
 - ▶ /boot/config-<version>
Configuración del Kernel para esa versión
- ▶ Por lo general, vuelve a ejecutar la utilidad de configuración del gestor de arranque para tomar el nuevo kernel en cuenta.



- ▶ make modules_install
 - ▶ Realiza por defecto la instalación para el sistema anfitrión, por lo tanto necesita ejecutarse como root
- ▶ Instala todos los módulos en /lib/modules/<version>/
 - ▶ kernel/
Los archivos de módulo .ko (Kernel Object), con la misma estructura de directorios que los fuentes.
 - ▶ modules.alias
Alias del módulo para las utilidades de carga del módulo. Línea de ejemplo:
alias sound-service-?-0 snd_mixer_oss
 - ▶ modules.dep
Dependencia de módulos
 - ▶ modules.symbols
Indica a qué módulo pertenece un símbolo dado.



Objetivos de limpieza del Kernel

- ▶ Limpia los archivos generados (para forzar la recompilación):
`make clean`
- ▶ Elimina todos los archivos generados. Necesario cuando se cambia de una arquitectura a otra. Precaución: también elimina su archivo `.config`!
`make mrproper`
- ▶ También elimina los archivos de copia de seguridad del editor (editor backup) y rechazo de parches (patch reject) (principalmente para generar parches):
`make distclean`





Compilación cruzada del núcleo



Cuando compila un kernel de Linux para otra arquitectura de CPU

- ▶ Mucho más rápido que compilar de forma nativa, cuando el sistema de destino es mucho más lento que su estación de trabajo GNU/Linux
- ▶ Mucho más fácil ya que las herramientas de desarrollo para su estación de trabajo GNU/Linux son mucho más fáciles de encontrar
- ▶ Para marcar la diferencia con un compilador nativo, los ejecutables de compilación cruzada tienen como prefijo el nombre del sistema de destino, la arquitectura y, a veces, la biblioteca. Ejemplos:

`mips-linux-gcc`, el prefijo es `mips-linux-`

`arm-linux-gnueabi-gcc`, el prefijo es `arm-linux-gnueabi-`



La arquitectura de la CPU y el prefijo de compilador cruzado se definen a través de las variables `ARCH` y `CROSS_COMPILE` en el Makefile de nivel superior.

- ▶ `ARCH` es el nombre de la arquitectura. Se define por el nombre del subdirectorio en `arch/` en las fuentes del núcleo
 - ▶ Ejemplo: `arm` si desea compilar un núcleo para la arquitectura `arm`
- ▶ `CROSS_COMPILE` es el prefijo de las herramientas de compilación cruzada
 - ▶ Ejemplo: `arm-linux-` si su compilador es `arm-linux-gcc`



Dos soluciones para definir ARCH y CROSS_COMPILE:

- ▶ Pase ARCH y CROSS_COMPILE en la línea de comando `make`:

```
make ARCH=arm CROSS_COMPILE=arm-linux- ...
```

Inconveniente: es fácil olvidar pasar estas variables cuando ejecuta cualquier comando `make`, lo que hace que su compilación y configuración se arruinen

- ▶ Defina ARCH y CROSS_COMPILE como variables de entorno: `export ARCH=arm`
`export CROSS_COMPILE=arm-linux-`

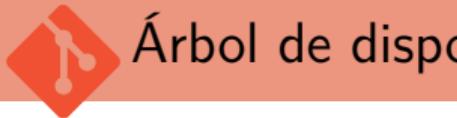
Inconveniente: solo funciona dentro del shell o terminal actual. Puede poner esta configuración en un archivo que obtenga cada vez que comience a trabajar en el proyecto. Si solo trabaja en una sola arquitectura con siempre la misma cadena de herramientas, incluso podría poner esta configuración en su archivo `~/.bashrc` para hacerlos permanentes y visibles desde cualquier terminal.



- ▶ Archivos de configuración predeterminados disponibles, por placa o por familia de CPU
 - ▶ Se almacenan en `arch/<arch>/configs/`, y son solo archivos mínimos de `.config`
 - ▶ Esta es la forma más común de configurar un núcleo para plataformas integradas
- ▶ Ejecute `make help` para encontrar si hay alguno disponible para su plataforma
- ▶ Para cargar un archivo de configuración predeterminado, simplemente ejecute `make acme_defconfig`
 - ▶ ¡Esto sobrescribirá su archivo `.config` existente!
- ▶ Para crear su propio archivo de configuración predeterminado
 - ▶ `make savedefconfig`, para crear un archivo de configuración mínimo
 - ▶ `mv defconfig arch/<arch>/configs/myown_defconfig`



- ▶ Después de cargar un archivo de configuración predeterminado, puede ajustar la configuración a sus necesidades con las interfaces normales `xconfig`, `gconfig` o `menuconfig`
- ▶ También puede iniciar la configuración desde cero sin cargar un archivo de configuración predeterminado
- ▶ Como la arquitectura es diferente de la arquitectura de tu host
 - ▶ Algunas opciones serán diferentes de la configuración nativa (opciones específicas de procesador y arquitectura, controladores específicos, etc.)
 - ▶ Muchas opciones serán idénticas (sistemas de archivos, protocolos de red, controladores independientes de la arquitectura, etc.)
- ▶ Asegúrese de tener el soporte para la CPU correcta, la placa correcta y los controladores de dispositivo correctos.



Árbol de dispositivos

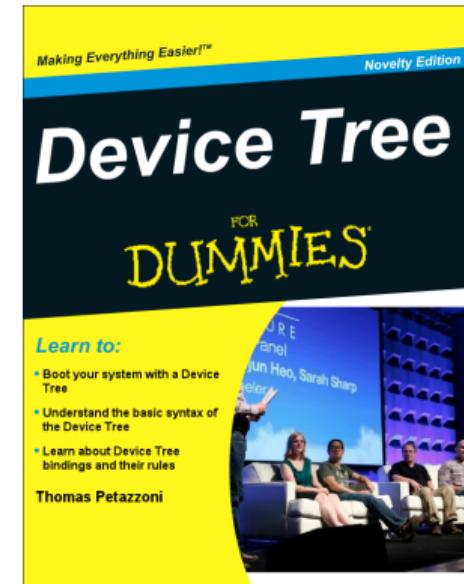
- ▶ Muchas arquitecturas integradas tienen una gran cantidad de hardware no reconocible.
- ▶ Dependiendo de la arquitectura, dicho hardware se describe usando un código C directamente dentro del núcleo o usando un lenguaje de descripción de hardware especial en un *Device Tree*.
- ▶ ARM, PowerPC, OpenRISC, ARC, Microblaze son ejemplos de arquitecturas que utilizan el Árbol de dispositivos.
- ▶ Un *Device Tree Source*, escrito por los desarrolladores del kernel, se compila en un binario *Device Tree Blob*, que se pasa al kernel en el momento del arranque.
 - ▶ Hay un árbol de dispositivos diferente para cada placa/plataforma compatible con el núcleo, disponible en `arch/arm/boot/dts/<board>.dtb`.
- ▶ El gestor de arranque debe cargar la imagen del kernel y el Blob del árbol de dispositivos en la memoria antes de iniciar el kernel.



¡Personaliza tu árbol de dispositivo de placa!

A menudo necesario para usuarios de placas integradas:

- ▶ Describir dispositivos externos conectados a buses no detectables (como I2C) y configurarlos.
- ▶ Para configurar el pin muxing: elegir qué señales de SoC están disponibles en los conectores externos de la placa.
- ▶ Para configurar algunos parámetros del sistema: particiones flash, línea de comando del kernel (existen otras formas)
- ▶ Referencia útil: Árbol de dispositivos para tontos, Thomas Petazzoni (abril de 2014):
<http://j.mp/1jQU6NR>





Construyendo e instalando el kernel

- ▶ Ejecute `make`
- ▶ Copie la imagen final del kernel en el almacenamiento de destino
 - ▶ puede ser `uImage`, `zImage`, `vmlinu`x, `bzImage` en `arch/<arch>/boot`
 - ▶ También podría ser necesario copiar el Blob de árbol de dispositivos, están disponibles en `arch/<arch>/boot/dts`
- ▶ `make install` rara vez se usa en el desarrollo integrado, ya que la imagen del núcleo es un archivo único, fácil de manejar
 - ▶ Sin embargo, es posible personalizar el comportamiento `make install` en `arch/<arch>/boot/install.sh`
- ▶ `make modules_install` se usa incluso en el desarrollo integrado, ya que instala muchos módulos y archivos de descripción
 - ▶ `make INSTALL_MOD_PATH=<dir>/ modules_install`
 - ▶ La variable `INSTALL_MOD_PATH` es necesaria para instalar los módulos en el sistema de archivos raíz de destino en lugar de su sistema de archivos raíz host.



- ▶ Las versiones recientes de U-Boot pueden arrancar el binario `zImage`.
- ▶ Las versiones anteriores requieren un formato especial de imagen del núcleo: `uImage`
 - ▶ `uImage` se genera a partir de `zImage` utilizando la herramienta `mkimage`. Se realiza automáticamente por el destino kernel `make uImage`.
 - ▶ En algunas plataformas ARM, `make uImage` requiere pasar una variable de entorno `LOADADDR`, que indica en qué dirección de memoria física se ejecutará el núcleo.
- ▶ Además de la imagen del núcleo, U-Boot también puede pasar un *Device Tree Blob* al núcleo.
- ▶ El proceso de arranque típico es por lo tanto:
 1. Cargue `zImage` o `uImage` en la dirección X en la memoria
 2. Cargue `<board>.dtb` en la dirección Y en la memoria
 3. Inicie el kernel con `bootz X - Y` (usando `zImage`), o `bootm X - Y` (usando `uImage`)
El `-` en el medio indica que no hay *initramfs*



- ▶ Además de la configuración del tiempo de compilación, el comportamiento del núcleo se puede ajustar sin recompilación utilizando la línea de comando **kernel**
- ▶ La línea de comando del núcleo es una cadena que define varios argumentos para el núcleo
 - ▶ Es muy importante para la configuración del sistema
 - ▶ `root=` para el sistema de archivos raíz (cubierto más adelante)
 - ▶ `console=` para el destino de los mensajes del kernel
 - ▶ Existen muchos más. Los más importantes están documentados en `admin-guide/kernel-parameter` en la documentación del kernel.
- ▶ Esta línea de comando del núcleo es
 - ▶ Pasado por el gestor de arranque. En U-Boot, el contenido de la variable de entorno `bootargs` se pasa automáticamente al núcleo
 - ▶ Especificado en el Árbol de dispositivos (para arquitecturas que lo usan)
 - ▶ Integrado en el kernel, utilizando la opción `CONFIG_CMDLINE`.



Laboratorio práctico - Kernel de compilación cruzada



- ▶ Configurar el entorno de compilación cruzada
- ▶ Configure y compile de forma cruzada el kernel para una plataforma `arm`
- ▶ En esta plataforma, interactúe con el gestor de arranque y arranque su núcleo



Usando módulos del núcleo



- ▶ Los módulos facilitan el desarrollo de controladores sin reiniciar: cargar, probar, descargar, reconstruir, cargar ...
- ▶ Útil para mantener el tamaño de la imagen del núcleo al mínimo (esencial en las distribuciones de GNU/Linux para PC).
- ▶ También es útil para reducir el tiempo de arranque: no pasa tiempo inicializando dispositivos y funciones del núcleo que solo necesitará más adelante.
- ▶ Precaución: una vez cargado, tenga pleno control y privilegios en el sistema. Sin protección particular. Es por eso que solo el usuario `root` puede cargar y descargar módulos.



- ▶ Algunos módulos del núcleo pueden depender de otros módulos, que deben cargarse primero
 - ▶ Ejemplo: el módulo `ubifs` depende de los módulos `ubi` y `mtd`.
 - ▶ Las dependencias se describen tanto en
 - `/lib/modules/<kernel-version>/modules.dep` como en
 - `/lib/modules/<kernel-version>/modules.dep.bin`
- Estos archivos se generan cuando ejecuta `make modules_install`



Kernel log

Registro del núcleo Cuando se carga un nuevo módulo, la información relacionada está disponible en el registro del núcleo.

- ▶ El núcleo mantiene sus mensajes en un búfer circular (para que no consuma más memoria con muchos mensajes)...
- ▶ Los mensajes de registro del kernel están disponibles a través del comando `dmesg` (**diagnostic message**)
- ▶ Los mensajes de registro del kernel también se muestran en la consola del sistema (los mensajes de la consola se pueden filtrar por nivel usando el parámetro de línea de comando del kernel `loglevel`, o completamente deshabilitado con el parámetro `quiet`. Ejemplo:

```
console=ttyS0 root=/dev/mmcblk0p2 loglevel=5
```

- ▶ Tenga en cuenta que también puede escribir en el registro del núcleo desde el espacio del usuario:

```
echo "<n>Información de depuración"> /dev/kmsg
```



Utilidades de módulos (1)

<module_name>: nombre del archivo del módulo sin el .ko

- ▶ `modinfo <module_name>` (para módulos en /lib/modules)
`modinfo <module_path>.ko`

Obtiene información sobre un módulo sin cargarlo: parámetros, licencia, descripción y dependencias.

- ▶ `sudo insmod <module_path>.ko`
Intenta cargar el módulo dado. Se debe proporcionar la ruta completa al archivo de objeto del módulo.



Comprender los problemas de carga del módulo

- ▶ Cuando falla la carga de un módulo, `insmod` a menudo no da suficientes detalles.
- ▶ Los detalles a menudo están disponibles en el registro del kernel.
- ▶ 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 (2)

- ▶ `sudo modprobe <module_name>`

Uso más común de `modprobe`: intenta cargar todos los módulos de los que depende el módulo dado, y luego este módulo. Hay muchas otras opciones disponibles. `modprobe` busca automáticamente en `/lib/modules/<version>/` el archivo de objeto correspondiente al nombre del módulo dado

- ▶ `lsmod`

Muestra la lista de módulos cargados

¡Compare su salida con el contenido de `/proc/modules`!



Utilidades de módulos (3)

- ▶ `sudo rmmod <module_name>`
Intenta eliminar el módulo dado.
Solo se permitirá si el módulo ya no está en uso (por ejemplo, no hay más procesos que abran un archivo de dispositivo)
- ▶ `sudo modprobe -r <module_name>`
Intenta eliminar el módulo dado y todos los módulos dependientes (que ya no son necesarios después de eliminar el módulo)



Pasar parámetros a módulos

- ▶ Encuentra los parámetros disponibles:

```
modinfo usb-storage
```

- ▶ Mediante insmod:

```
sudo insmod ./usb-storage.ko delay_use=0
```

- ▶ Mediante modprobe:

Establezca los parámetros en `/etc/modprobe.conf` o en cualquier archivo en `/etc/modprobe.d/`:

- ▶ A través de la línea de comando del núcleo, cuando el controlador está integrado estáticamente en el núcleo:

```
usb-storage.delay_use=0
```

- ▶ `usb-storage` es el *nombre del driver*
- ▶ `delay_use` es el *nombre del parámetro del controlador*. Especifica un retraso antes de acceder a un dispositivo de almacenamiento USB (útil para dispositivos rotativos)
- ▶ `0` es el *valor del parámetro del driver*



Verifique los valores de los parámetros del módulo

¿Cómo encontrar/editar los valores actuales para los parámetros de un módulo cargado?

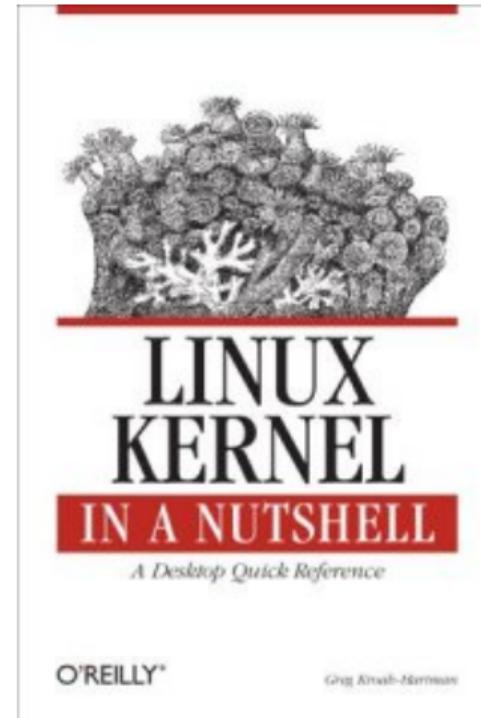
- ▶ Verifique `/sys/module/<name>/parameters`.
- ▶ Hay un archivo por parámetro, que contiene el valor del parámetro.
- ▶ También es posible cambiar los valores de los parámetros si estos archivos tienen permisos de escritura (depende del código del módulo)
- ▶ Ejemplo:

```
echo 0 > /sys/module/usb_storage/parameters/delay_use
```



Linux Kernel in a Nutshell, Dec. 2006

- ▶ Por Greg Kroah-Hartman, O'Reilly
<http://www.kroah.com/lkn/>
- ▶ Un buen libro de referencia y una guía sobre la configuración, compilación y administración de las fuentes del kernel de Linux
- ▶ ¡Disponible gratis en línea!
¡Gran compañero del libro impreso para búsquedas electrónicas fáciles!
Disponible como archivo PDF único en
<https://bootlin.com/community/kernel/lkn/>
- ▶ Envejecer pero aún contener contenido útil





Device Tree



Visión general

- ▶ Es una estructura de datos para describir al Hardware
- ▶ Se pasa al Kernel en el momento de inicio
- ▶ Es una alternativa a los detalles de la plataforma codificadas en el propio kernel
- ▶ Tanto el Device Tree (DT) como el Device Tree Overlay (DTO) son una forma de describir el Hardware del sistema
- ▶ Un ejemplo de esto sería describir cómo las UART interactúan con el sistema, que pines, como deben ser multiplexadas, qué dispositivo habilitar y qué controlador utilizar.
- ▶ El BeagleBone original no usó el DT, pero el recientemente lanzado BeagleBone Black introdujo la descripción de Hardware utilizando DT con la versión del kernel de Linux 3.8.
- ▶ Las siguientes páginas intentarán desglosar los conceptos, y dar ejemplos de cómo y por qué querrías usar el device tree en el desarrollo diario



- ▶ Arbol de nodos y propiedades
 - ▶ Los nodos dan estructura
 - ▶ Las propiedades agregan detalle
 - ▶ Pares clave-valor
- ▶ propiedad 'compatible'
 - ▶ Cada valor 'compatible' asociado con un 'binding'
- ▶ 'phandles'
 - ▶ conexiones secundarias entre nodos
 - ▶ irqs, gpios, mdio, i2s, etc



Ejemplo de Árbol de Dispositivos

```
/ {  
    node1 {  
        a-string-property = "A string";  
        a-string-list-property = "first string", "second string";  
        a-byte-data-property = [0x01 0x23 0x34 0x56];  
        child-node1 {  
            first-child-property;  
            second-child-property = <1>;  
            a-string-property = "Hello, world";  
        };  
        child-node2 {  
        };  
    };  
    node2 {  
        an-empty-property;  
        a-cell-property = <1 2 3 4>; /* each number (cell) is a uint32 */  
        child-node1 {  
        };  
    };  
};
```



Dos soluciones para definir ARCH y CROSS_COMPILE:

- ▶ Pasamos ARCH y CROSS_COMPILE en el make línea de comando:

```
make ARCH=arm CROSS_COMPILE=arm-linux- ...
```

Inconveniente: es fácil olvidar pasar estas variables cuando ejecutamos cualquier comando `make`, lo que provoca que la compilación y la configuración se estropeen.

- ▶ Defina ARCH y CROSS_COMPILE como entorno variables:

```
export ARCH=arm
```

```
export CROSS_COMPILE=arm-linux-
```

Inconveniente: solo funciona dentro del terminal o shell actual. Podrías poner estas configuraciones en un archivo que se ejecute como fuente `. archivo` cada vez que empiezas a trabajar en el proyecto. Si solo trabajas en una arquitectura única siempre con el misma juego de herramientas, incluso podría poner estas configuraciones en su `~/.bashrc` para que sean permanentes y visibles desde cualquier terminal.



Ejemplo 2

```
/*
 * Copyright (C) 2013 CircuitCo
 *
 * Virtual cape for UART1 on connector pins P9.24 P9.26
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "BB-UART1";
    version = "00A0";

    /* state the resources this cape uses */
    exclusive-use =
        /* the pin header uses */
        "P9.24", /* uart1_txd */
        "P9.26", /* uart1_rxd */
        /* the hardware ip uses */
        "uart1";
```



Ejemplo 2 cont

```
fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {
        bb_uart1_pins: pinmux_bb_uart1_pins {
            pinctrl-single,pins = <
                0x184 0x20 /* P9.24 uart1_txd.uart1_txd MODE0 OUTPUT (TX) */
                0x180 0x20 /* P9.26 uart1_rxd.uart1_rxd MODE0 INPUT (RX) */
            >;
        };
    };
};

fragment@1 {
    target = <&uart2>; /* really uart1 */
    __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart1_pins>;
    };
};
};
```



Archivos de configuración predefinidos

- ▶ Archivos de configuración predeterminados disponibles, por placa o por familia de CPU
 - ▶ Se almacenan en `arch/<arch>/configs/`, y son solo archivos de código mínimo `.config`
 - ▶ Esta es la forma más común de configurar un kernel para plataformas embebidas
- ▶ Ejecute `make help` para encontrar si hay alguno disponible para su plataforma
- ▶ Para cargar un archivo de configuración predeterminado, simplemente ejecute `make acme_defconfig`
 - ▶ ¡Esto sobrescribirá su archivo `.config` existente!
- ▶ Para crear su propio archivo de configuración por defecto
 - ▶ `make savedefconfig`, para crear un archivo mínimo de configuración
 - ▶ `mv defconfig arch/<arch>/configs/myown_defconfig`



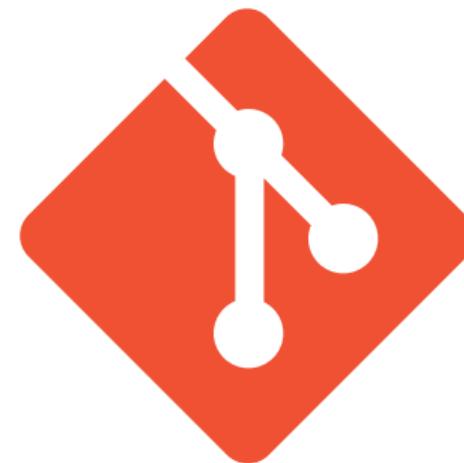
Sistema de archivos raíz de Linux

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Principio y soluciones



- ▶ Los sistemas de archivos se utilizan para organizar datos en directorios y archivos en dispositivos de almacenamiento o en la red. Los directorios y archivos están organizados como una jerarquía.
- ▶ En los sistemas Unix, las aplicaciones y los usuarios ven una **única jerarquía global** de archivos y directorios, que pueden estar compuestos por varios sistemas de archivos.
- ▶ Los sistemas de archivos están **montados** en una ubicación específica en esta jerarquía de directorios
 - ▶ Cuando se monta un sistema de archivos en un directorio (llamado *punto de montaje*), el contenido de este directorio refleja el contenido del dispositivo de almacenamiento
 - ▶ Cuando se desmonta el sistema de archivos, el *punto de montaje* está vacío nuevamente
- ▶ Esto permite que las aplicaciones accedan fácilmente a archivos y directorios, independientemente de su ubicación de almacenamiento exacta



- ▶ Cree un punto de montaje (es solo un directorio)

```
$ mkdir /mnt/usbkey
```

- ▶ Esta vacío

```
$ ls /mnt/usbkey  
$
```

- ▶ Monte un dispositivo de almacenamiento en este punto de montaje

```
$ sudo mount -t vfat /dev/sda1 /mnt/usbkey  
$
```

- ▶ Puede acceder al contenido del dispositivo USB

```
$ ls /mnt/usbkey  
docs prog.c picture.png movie.avi  
$
```



mount / umount

- ▶ `mount` permite montar sistemas de archivos
 - ▶ `mount -t type device mountpoint`
 - ▶ `type` es el tipo de sistema de archivos
 - ▶ `device` es el dispositivo de almacenamiento o la ubicación de red a montar
 - ▶ `mountpoint` es el directorio donde se podrá acceder a los archivos del dispositivo de almacenamiento o la ubicación de la red
 - ▶ `mount` sin argumentos muestra los sistemas de archivos montados actualmente
- ▶ `umount` permite desmontar sistemas de archivos
 - ▶ Esto es necesario antes de reiniciar, o antes de desconectar un Pendrive USB, porque el caché del kernel de Linux escribe en la memoria para aumentar el rendimiento.
`umount` se asegura de que estas escrituras se confirmen en el almacenamiento



- ▶ Un sistema de archivos particular se monta en la raíz de la jerarquía, identificado por /
- ▶ Este sistema de archivos se denomina **root filesystem**
- ▶ Como `mount` y `umount` son programas, son archivos dentro de un sistema de archivos
 - ▶ No son accesibles antes de montar al menos un sistema de archivos.
- ▶ Como el sistema de archivos raíz es el primer sistema de archivos montado, no se puede montar con el comando normal `mount`
- ▶ Está montado directamente por el kernel, de acuerdo con la opción de kernel `root=`
- ▶ Cuando no hay un sistema de archivos raíz disponible, el núcleo entra en pánico

```
Please append a correct "root=" boot option
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown block(0,0)
```



Ubicación del sistema de archivos raíz

- ▶ Se puede montar desde diferentes lugares
 - ▶ Desde la partición de un disco rígido
 - ▶ Desde la partición de un Pendrive USB
 - ▶ Desde la partición de una tarjeta SD
 - ▶ Desde la partición de un chip flash NAND o un tipo similar de dispositivo de almacenamiento
 - ▶ Desde la red, usando el protocolo NFS
 - ▶ Desde la memoria, utilizando un sistema de archivos precargado (por el gestor de arranque)
 - ▶ etc.
- ▶ Depende del diseñador del sistema elegir la configuración del sistema y configurar el comportamiento del núcleo con `root=`

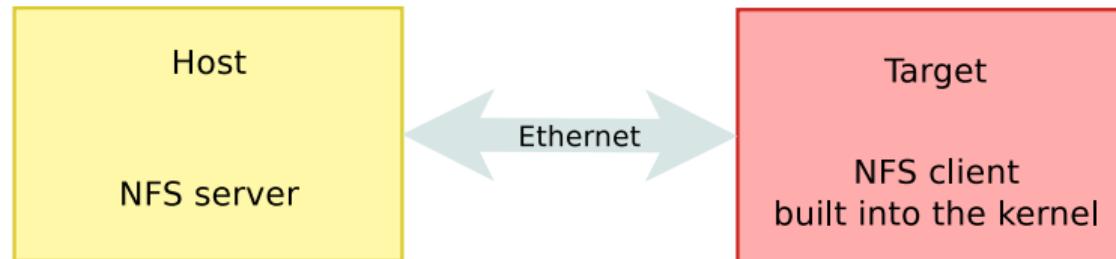
- ▶ Particiones de un disco rígido o Pendrive USB
 - ▶ `root=/dev/sdXY`, donde X es una letra que indica el dispositivo, y Y un número que indica la partición
 - ▶ `/dev/sdb2` es la segunda partición de la segunda unidad de disco (Pendrive USB o disco rígido ATA)
- ▶ Particiones de una tarjeta SD
 - ▶ `root=/dev/mmcblkXpY`, donde X es un número que indica el dispositivo y Y un número que indica la partición
 - ▶ `/dev/mmcblk0p2` es la segunda partición del primer dispositivo
- ▶ Particiones de almacenamiento flash NAND
 - ▶ `root=/dev/mtdblockX`, donde X es el número de partición
 - ▶ `/dev/mtdblock3` es la cuarta partición de un chip flash NAND (si solo está presente un chip flash NAND)



Montaje de rootfs a través de la red 1/4

Una vez que la red funciona, su sistema de archivos raíz podría ser un directorio en su host de desarrollo GNU/Linux, exportado por NFS (Network File System). Esto es muy conveniente para el desarrollo del sistema:

- ▶ Facilita la actualización de archivos en el sistema de archivos raíz, sin necesidad de reiniciar. Mucho más rápido que a través del puerto serie.
- ▶ Puede tener un gran sistema de archivos raíz incluso si aún no tiene soporte para almacenamiento interno o externo.
- ▶ El sistema de archivos raíz puede ser enorme. Incluso puede compilar herramientas de compilación nativas y compilar todas las herramientas que necesita en el propio destino (aunque es mejor realizar una compilación cruzada).





Montaje de rootfs a través de la red 2/4

En el lado de la estación de trabajo de desarrollo, se necesita un servidor NFS

- ▶ Instalar un servidor NFS (ejemplo: Debian, Ubuntu)

```
sudo apt-get install nfs-kernel-server
```

- ▶ Agregue el directorio exportado a su archivo /etc/exports:

```
/home/tux/rootfs 192.168.1.111(rw,no_root_squash,no_subtree_check)
```

- ▶ 192.168.1.111 es la dirección IP del cliente

- ▶ rw,no_root_squash,no_subtree_check son las opciones del servidor NFS para esta exportación de directorio

- ▶ Inicie o reinicie su servidor NFS (ejemplo: Debian, Ubuntu)

```
sudo /etc/init.d/nfs-kernel-server restart
```

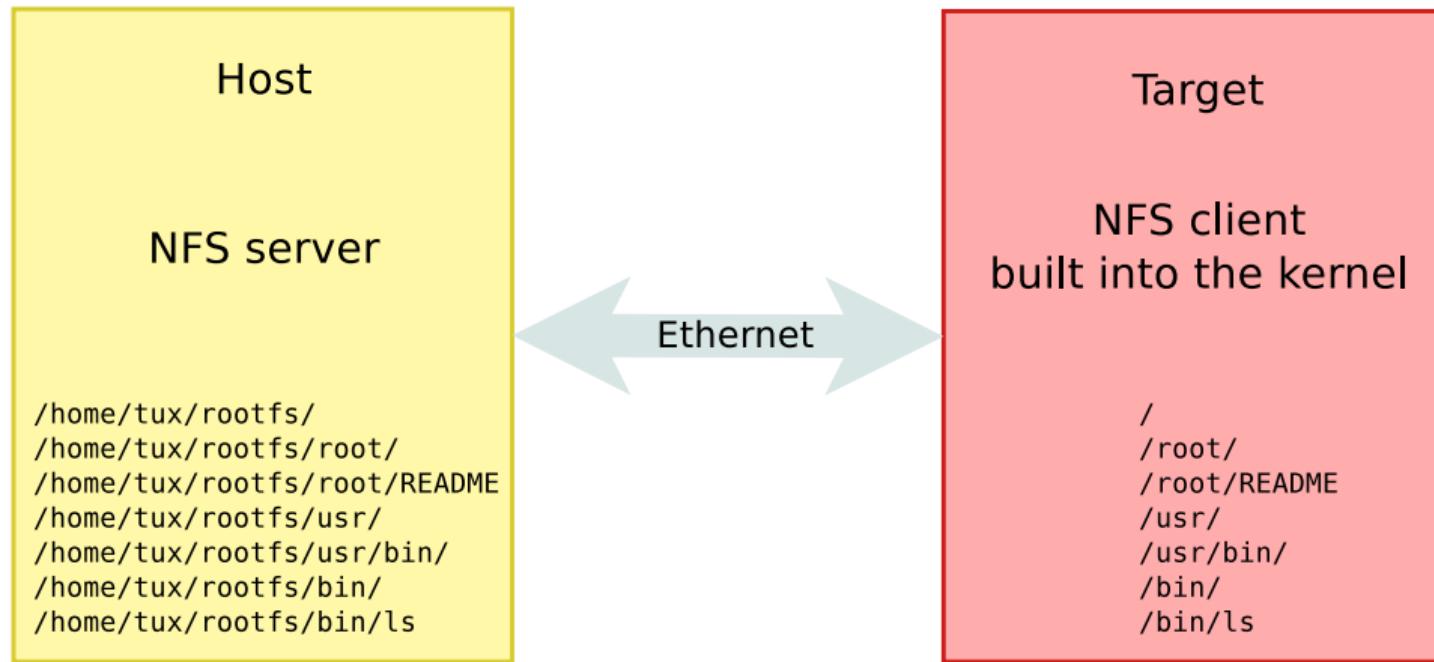


Montaje de rootfs a través de la red 3/4

- ▶ En el sistema de destino
- ▶ El núcleo debe compilarse con
 - ▶ CONFIG_NFS_FS=y (Soporte NFS)
 - ▶ CONFIG_IP_PNP=y (configurar IP en el momento del arranque)
 - ▶ CONFIG_ROOT_NFS=y (soporte para NFS como rootfs)
- ▶ El kernel debe iniciarse con los siguientes parámetros:
 - ▶ root=/dev/nfs (queremos rootfs sobre NFS)
 - ▶ ip=192.168.1.111 (dirección IP de destino)
 - ▶ nfsroot=192.168.1.110:/home/tux/rootfs/ (Detalles del servidor NFS)



Montaje de rootfs a través de la red 4/4





rootfs en memoria: initramfs 1/3

- ▶ También es posible tener el sistema de archivos raíz integrado en la imagen del kernel
- ▶ Por lo tanto, se carga en la memoria junto con el núcleo
- ▶ Este mecanismo se llama **initramfs**
 - ▶ Integra un archivo comprimido del sistema de archivos en la imagen del kernel
 - ▶ Variante: el cargador comprimido también puede cargar el archivo comprimido por separado.
- ▶ Es útil para dos casos
 - ▶ Arranque rápido de sistemas de archivos raíz muy pequeños. Como el sistema de archivos está completamente cargado en el momento del arranque, el inicio de la aplicación es muy rápido.
 - ▶ Como paso intermedio antes de cambiar a un sistema de archivos raíz real, ubicado en dispositivos para los que se necesitan controladores que no forman parte de la imagen del kernel (controladores de almacenamiento, controladores de sistemas de archivos, controladores de red). Esto siempre se utiliza en el núcleo de las distribuciones de escritorio/servidor para mantener el tamaño de la imagen del núcleo razonable.



Código del Kernel y datos

Sistema de archivos raíz
almacenado como archivo
cpio comprimido

Imagen del Kernel (vmlinux, bzImage, etc.)



- ▶ El contenido de un initramfs se define en el nivel de configuración del kernel, con la opción `CONFIG_INITRAMFS_SOURCE`
 - ▶ Puede ser la ruta a un directorio que contiene el contenido del sistema de archivos raíz.
 - ▶ Puede ser la ruta a un archivo cpio
 - ▶ Puede ser un archivo de texto que describa el contenido de initramfs
(consulte la documentación para obtener más detalles)
- ▶ El proceso de compilación del kernel tomará automáticamente el contenido de la opción `CONFIG_INITRAMFS_SOURCE` e integrará el sistema de archivos raíz en la imagen del kernel
- ▶ Detalles (en los fuentes del kernel):
`filesystems/ramfs-rootfs-initramfs.txt`
`early-userspace/README`



Contenido del sistema de archivos raíz



Organización del sistema de archivos raíz

- ▶ La organización de un sistema de archivos raíz de Linux en términos de directorios está bien definida por el **Estándar de Jerarquía del Sistema de Archivos**
- ▶ <http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs>
- ▶ La mayoría de los sistemas Linux cumplen con esta especificación
 - ▶ Las aplicaciones esperan esta organización
 - ▶ Facilita a los desarrolladores y usuarios, ya que la organización del sistema de archivos es similar en todos los sistemas



Directorios importantes 1/2

- /bin Programas basicos
- /boot Imagen del núcleo (solo cuando el núcleo se carga desde un sistema de archivos, no es común en arquitecturas que no son x86)
- /dev Archivos de dispositivo (cubiertos más adelante)
- /etc Configuración de todo el sistema
- /home Directorio para los directorios de inicio de los usuarios
 - /lib Bibliotecas básicas
- /media Puntos de montaje para medios extraíbles
- /mnt Puntos de montaje para medios estáticos
- /proc Punto de montaje para el sistema de archivos virtual proc



Directorios importantes 2/2

/root Directorio de inicio del usuario root

/sbin Programas básicos del sistema

/sys Punto de montaje del sistema de archivos virtual sysfs

/tmp Archivos temporales

/usr /usr/bin Programas no basicos

 /usr/lib Bibliotecas no básicas

 /usr/sbin Programas del sistema no básicos

/var Archivos de datos variables. Esto incluye directorios y archivos de spool, datos administrativos y de registro, y archivos transitorios y temporales



- ▶ Los programas básicos se instalan en `/bin` y `/sbin` y las bibliotecas básicas en `/lib`
- ▶ Todos los demás programas se instalan en `/usr/bin` y `/usr/sbin` y todas las demás bibliotecas en `/usr/lib`
- ▶ En el pasado, en sistemas Unix, `/usr` se montaba muy a menudo en la red, a través de NFS
- ▶ Para permitir que el sistema arranque cuando la red estaba inactiva, algunos binarios y bibliotecas se almacenaban en `/bin`, `/sbin` y `/lib`
- ▶ `/bin` y `/sbin` contienen programas como `ls`, `ifconfig`, `cp`, `bash`, etc
- ▶ `/lib` contiene la biblioteca C y algunas veces algunas otras bibliotecas básicas
- ▶ Todos los demás programas y bibliotecas están en `/usr`



Archivos de dispositivo



- ▶ Una de las funciones importantes del núcleo es **permitir que las aplicaciones accedan a dispositivos de hardware**
- ▶ En el kernel de Linux, la mayoría de los dispositivos se presentan a las aplicaciones de espacio de usuario a través de dos abstracciones diferentes
 - ▶ Dispositivo de **Caracter**
 - ▶ Dispositivo de **Bloques**
- ▶ Internamente, el núcleo identifica cada dispositivo mediante un triplete de información
 - ▶ **Tipo** (caracter o bloque)
 - ▶ **Mayor** (típicamente la categoría de dispositivo)
 - ▶ **Menor** (normalmente el identificador del dispositivo)



- ▶ Dispositivos de bloques
 - ▶ Un dispositivo compuesto por bloques de tamaño fijo, que se pueden leer y escribir para almacenar datos
 - ▶ Se utiliza para discos rígidos, Pendrives USB, tarjetas SD, etc
- ▶ Dispositivos de caracteres
 - ▶ Originalmente, una secuencia infinita de bytes, sin principio, sin fin, sin tamaño. El ejemplo puro: un puerto serie.
 - ▶ Se utiliza para puertos serie, terminales, pero también tarjetas de sonido, dispositivos de adquisición de video, frame buffers
 - ▶ La mayoría de los dispositivos que no son dispositivos de bloque están representados como dispositivos de caracteres por el kernel de Linux



Pseudo sistemas de archivos



- ▶ El sistema de archivos virtual `proc` existe desde el comienzo de Linux
- ▶ Permite
 - ▶ El núcleo para exponer estadísticas sobre procesos en ejecución en el sistema
 - ▶ El usuario debe ajustar en tiempo de ejecución varios parámetros del sistema sobre gestión de procesos, gestión de memoria, etc.
- ▶ El sistema de archivos `proc` es utilizado por muchas aplicaciones de espacio de usuario estándar, y esperan que se monte en `/proc`
- ▶ Las aplicaciones como `ps` o `top` no funcionarían sin el sistema de archivos `proc`
- ▶ Comando para montar `/proc`:
`mount -t proc nodev /proc`
- ▶ `filesystems/proc.txt` en las fuentes del kernel
- ▶ `man proc`



- ▶ Un directorio para cada proceso en ejecución en el sistema
 - ▶ `/proc/<pid>`
 - ▶ `cat /proc/3840/cmdline`
 - ▶ Contiene detalles sobre los archivos abiertos por el proceso, el uso de la CPU y la memoria, etc.
- ▶ `/proc/interrupts`, `/proc/devices`, `/proc/iomem`, `/proc/ioports` contiene información general relacionada con el dispositivo
- ▶ `/proc/cmdline` contiene la línea de comando del núcleo
- ▶ `/proc/sys` contiene muchos archivos en los que se puede escribir para ajustar los parámetros del kernel
 - ▶ Se llaman *sysctl*. Ver `sysctl/` en las fuentes del kernel.
 - ▶ Ejemplo
 - `echo 3 > /proc/sys/vm/drop_caches`



- ▶ El sistema de archivos `sysfs` es una característica integrada en el kernel de Linux 2.6
- ▶ Permite representar en el espacio del usuario la visión que el núcleo tiene de los buses, dispositivos y controladores del sistema.
- ▶ Es útil para varias aplicaciones de espacio de usuario que necesitan enumerar y consultar el hardware disponible, por ejemplo `udev` o `mdev`.
- ▶ Todas las aplicaciones que usan `sysfs` esperan que se monte en el directorio `/sys`
- ▶ Comando para montar `/sys`:
`mount -t sysfs nodev /sys`
- ▶ `$ ls /sys/`
`block bus class dev devices firmware`
`fs kernel module power`



Sistema de archivos mínimo

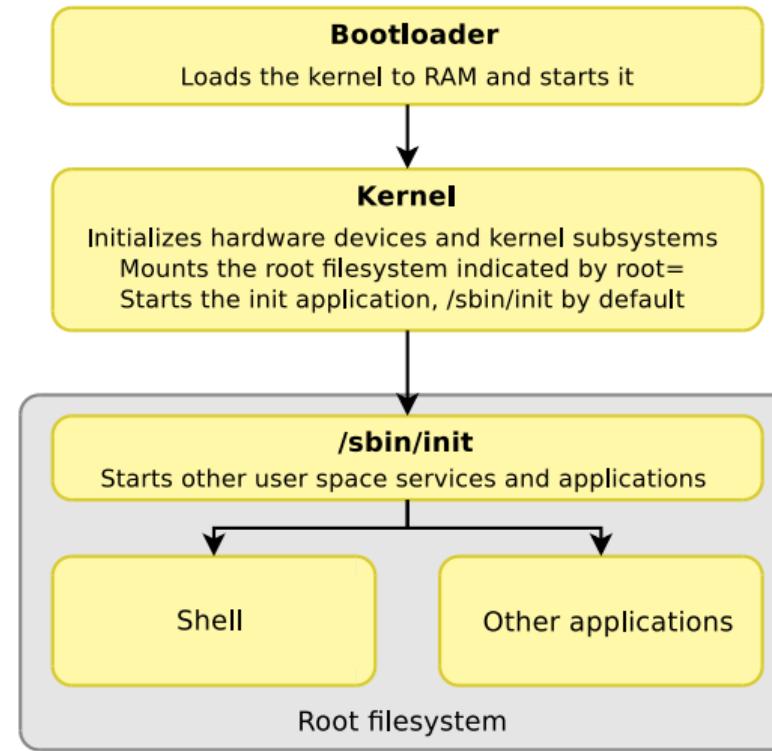


Aplicaciones basicas

- ▶ Para funcionar, un sistema Linux necesita al menos algunas aplicaciones
- ▶ Una aplicación `init`, que es la primera aplicación de espacio de usuario iniciada por el núcleo después de montar el sistema de archivos raíz
 - ▶ El núcleo intenta ejecutar `/sbin/init`, `/bin/init`, `/etc/init` y `/bin/sh`.
 - ▶ En el caso de un initramfs, solo buscará `/init`. Otra ruta puede ser proporcionada por el argumento del kernel `rdinit`.
 - ▶ Si no se encuentra ninguno de ellos, el núcleo entra en pánico y el proceso de arranque se detiene.
 - ▶ La aplicación `init` es responsable de iniciar todas las demás aplicaciones y servicios de espacio de usuario
- ▶ Generalmente un shell, para permitir que un usuario interactúe con el sistema
- ▶ Aplicaciones básicas de Unix, para copiar archivos, mover archivos, enumerar archivos (comandos como `mv`, `cp`, `mkdir`, `cat`, etc.)
- ▶ Estos componentes básicos deben integrarse en el sistema de archivos raíz para que sea utilizable

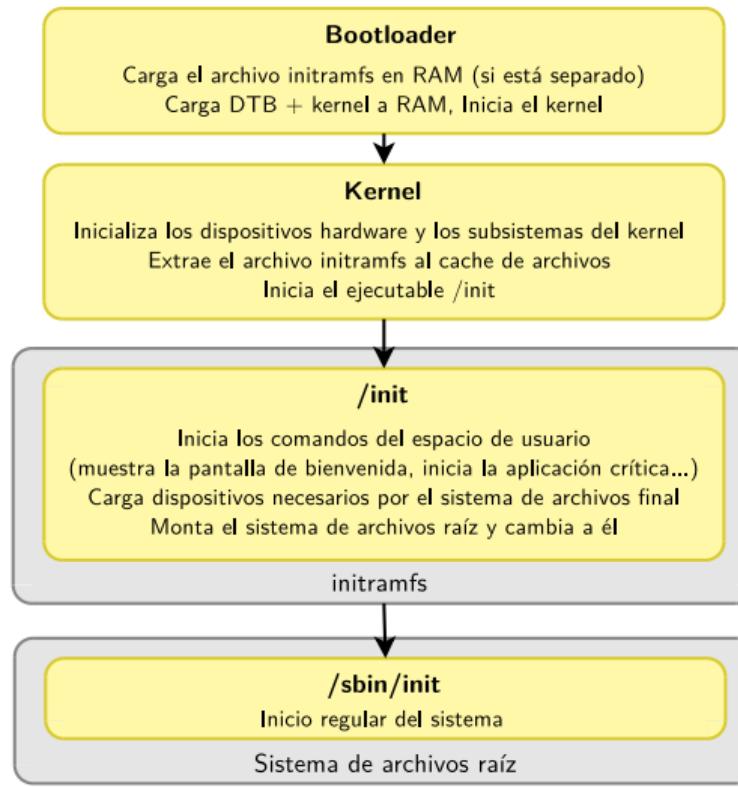


Proceso general de arranque





Proceso general de arranque con initramfs

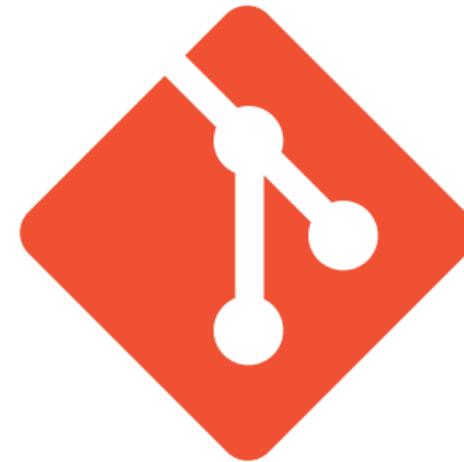




Busybox

Busybox

Free Electrons



© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!



¿Porqué Busybox?

- ▶ Un sistema Linux necesita un conjunto básico de programas para funcionar
 - ▶ Un programa init
 - ▶ Un shell
 - ▶ Varias utilidades básicas para la manipulación de archivos y la configuración del sistema.
- ▶ En sistemas Linux normales, estos programas son proporcionados por diferentes proyectos
 - ▶ coreutils, bash, grep, sed, tar, wget, modutils, etc. son todos proyectos diferentes
 - ▶ Muchos componentes diferentes para integrar
 - ▶ Componentes no diseñados teniendo en cuenta las limitaciones de los sistemas embebidos: no son muy configurables y tienen una amplia gama de características
- ▶ Busybox es una solución alternativa, extremadamente común en sistemas embebidos



Caja de herramientas de uso general: BusyBox

- ▶ Reescribe muchas utilidades útiles de línea de comandos de Unix
 - ▶ Integrado en un solo proyecto, lo que facilita trabajar con él
 - ▶ Diseñado teniendo en cuenta los sistemas integrados: altamente configurable, sin características innecesarias
- ▶ Todas las utilidades se compilan en un solo ejecutable, `/bin/busybox`
 - ▶ Se crean enlaces simbólicos a `/bin/busybox` para cada aplicación integrada en Busybox
- ▶ Para una configuración bastante funcional, menos de 500 KB (compilado estáticamente con uClibc) o menos de 1 MB (compilado estáticamente con glibc)
- ▶ <http://www.busybox.net/>



Comandos BusyBox!

Comandos disponibles en BusyBox 1.13

[, [[], addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, basename, bbconfig, bbsht, brctl, bunzip2, busybox, bzcat, bzip2, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devfsd, df, dhcprelay, diff, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg_deb, du, dumpkmap, dumpleases, e2fsck, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, find, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftpget, ftpput, fuser, getenforce, getopt, getsebool, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd, insmod, install, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, lash, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lzmacat, makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mknod, mkswap, mktemp, modprobe, more, mount, mountpoint, msh, mt, mv, nameif, nc, netstat, nice, nmeter, nohup, nslookup, od, openvt, parse, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, poweroff, printenv, printf, ps, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script, sed, selinuxenabled, sendmail, seq, sestatus, setarch, setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid, setuidgid, sh, sha1sum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon, stat, strings, stty, su, slogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, telnedt, test, tftp, tftpd, time, top, touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc, udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip]



Applt destacada: init de Busybox

- ▶ Busybox proporciona una implementación de un programa `init`
- ▶ Más simple que la implementación `init` que se encuentra en los sistemas de escritorio/servidor: no se implementan niveles de ejecución
- ▶ Un único archivo de configuración: `/etc/inittab`
 - ▶ Cada línea tiene la forma `<id>::<action>:<process>`
- ▶ Permite ejecutar servicios al inicio y asegurarse de que ciertos servicios siempre se ejecutan en el sistema
- ▶ Consulte `examples/inittab` en Busybox para obtener detalles sobre la configuración



- ▶ Si está utilizando BusyBox, agregar soporte `vi` solo agrega 20K. (construido con bibliotecas compartidas, usando uClibc).
- ▶ Puede seleccionar en qué características exactas compilar
- ▶ ¡Los usuarios apenas se dan cuenta de que están usando una versión ligera de `vi`!
- ▶ Consejo: puede aprender `vi` en el escritorio, ejecutando el comando `vimtutor`.



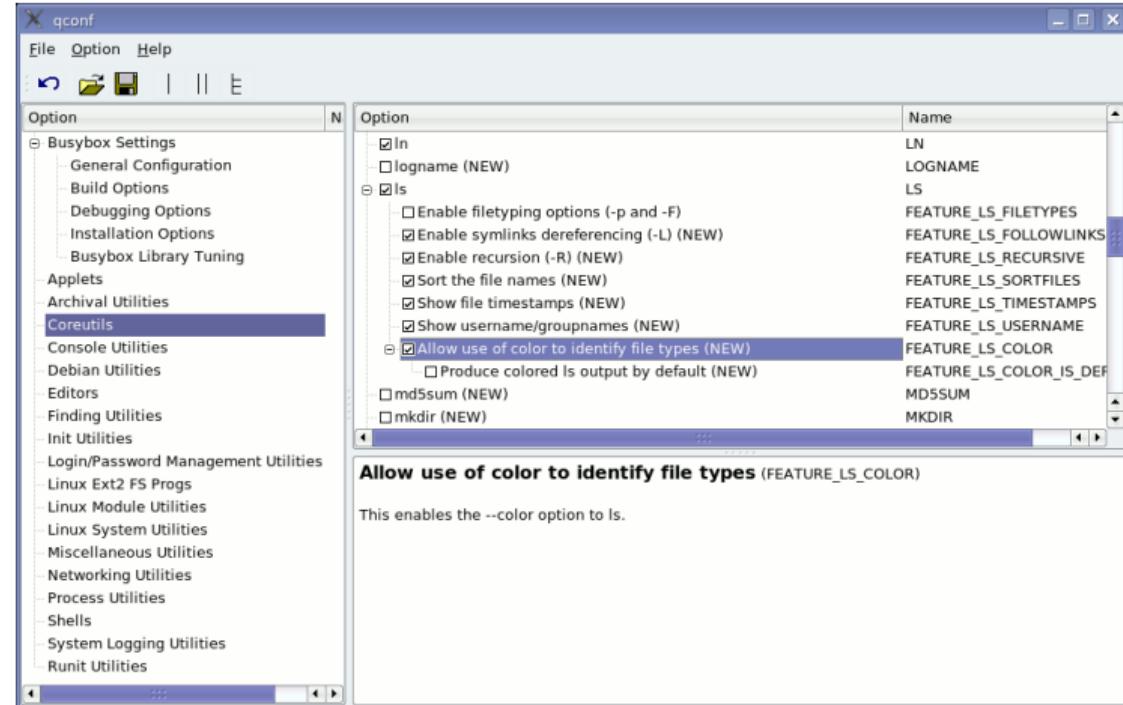
- ▶ Obtenga las últimas fuentes estables de <http://busybox.net>
- ▶ Configurar BusyBox (crea un archivo .config):
 - ▶ `make defconfig`
Es bueno comenzar con BusyBox.
Configura BusyBox con todas las opciones para usuarios habituales.
 - ▶ `make allnoconfig`
Anula la selección de todas las opciones. Bueno para configurar solo lo que necesitas.
- ▶ `make xconfig` (gráfico, necesita el paquete `libqt3-mt-dev`)
o `make menuconfig` (texto)
 - i Las mismas interfaces de configuración que las utilizadas por el kernel de Linux (aunque se utilizan versiones anteriores).



BusyBox make xconfig

Puedes elegir:

- ▶ los comandos para compilar,
- ▶ incluso las opciones y funciones de comando que necesita!





- ▶ Establezca el prefijo del compilador cruzado en la interfaz de configuración:
Settings → Build Options → Cross compiler prefix
Ejemplo: arm-linux-
- ▶ Establecer el directorio de instalación en la interfaz de configuración:
BusyBox Settings → Installation Options -
> Destination path for 'make install'
- ▶ Agregue la ruta del compilador cruzado a la variable de entorno PATH:
`export PATH=/usr/xtools/arm-unknown-linux-uclibcgnueabi/bin:$PATH`
- ▶ Compilar BusyBox:
`make`
- ▶ Instálelo (esto crea enlaces simbólicos de estructura de directorio Unix al ejecutable `busybox`):
`make install`



- ▶ Realice el arranque de Linux en un directorio en su estación de trabajo, compartido por NFS
- ▶ Cree y configure un sistema embebido minimalista de Linux
- ▶ Instale y use BusyBox
- ▶ Inicie el sistema con `/sbin/init`
- ▶ Configure una interfaz Web simple
- ▶ Use bibliotecas compartidas



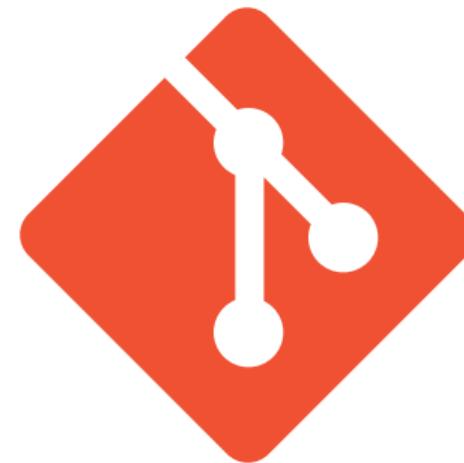
Sistemas de archivos en bloques

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Dispositivos en Bloque



- ▶ Los dispositivos de almacenamiento se clasifican en dos tipos principales:
dispositivos en bloques y **dispositivos flash**
 - ▶ Son manejados por diferentes subsistemas y diferentes sistemas de archivos
- ▶ **Dispositivos en bloques** se pueden leer y escribir por bloque, sin necesidad de borrar
 - ▶ Discos rígidos, disquetes, discos RAM
 - ▶ Pendrives USB, Compact Flash, tarjetas SD: se basan en el almacenamiento flash, pero tienen un controlador integrado que emula un dispositivo de bloque, administrando y borrando sectores flash de manera transparente
- ▶ **Dispositivos Flash**, se pueden leer, pero la escritura requiere borrado, y a menudo ocurre en un tamaño mayor que el tamaño de "bloque"
 - ▶ flash NOR, flash NAND



Lista de dispositivos en bloque

- ▶ La lista de todos los dispositivos de bloque disponibles en el sistema se puede encontrar en `/proc/partitions`

```
$ cat /proc/partitions
major minor #blocks name

179      0    3866624 mmcblk0
179      1     73712 mmcblk0p1
179      2    3792896 mmcblk0p2
 8       0   976762584 sda
 8       1   1060258 sda1
 8       2  975699742 sda2
```

- ▶ Y tambien en `/sys/block/`



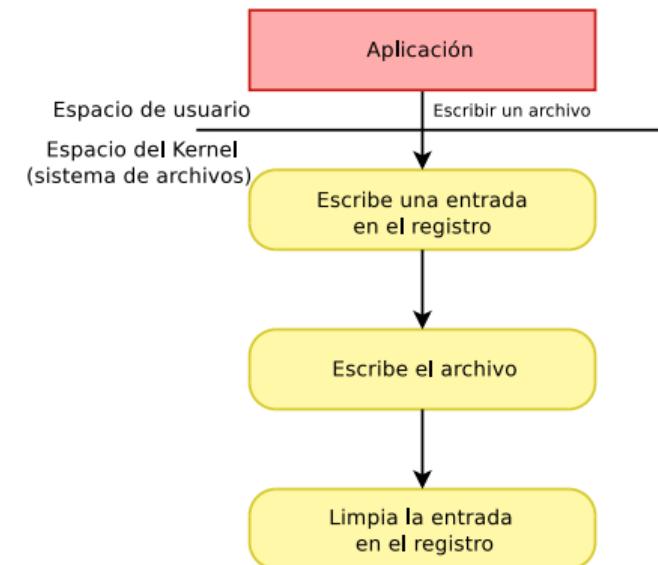
Sistemas de archivos tradicionales

- ▶ Pueden quedar en un estado no coherente después de un bloqueo del sistema o apagado repentino, lo que requiere una verificación completa del sistema de archivos después del reinicio
- ▶ ext2: sistema de archivos tradicional de Linux
(repararlo con `fsck.ext2`)
- ▶ vfat: sistema de archivos tradicional de Windows
(repararlo con `fsck.vfat` en GNU/Linux o Scandisk en Windows)

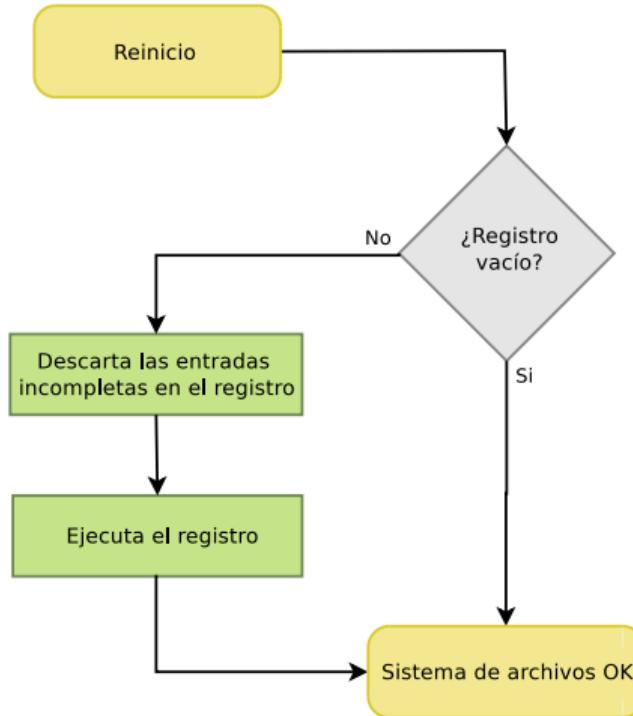


Sistemas de archivos con registro (Jounaled)

- ▶ Diseñado para permanecer en un estado correcto incluso después de fallas del sistema o un apagado repentino
- ▶ Todas las escrituras se describen por primera vez en el registro antes de escribir los archivos



Recuperación del sistema de archivos luego de una falla



- ▶ Gracias al registro, el sistema de archivos nunca se deja en un estado dañado
- ▶ Los datos guardados recientemente aún podrían perderse



Sistemas de archivos con registro

- ▶ ext3: ext2 con extensión para el registro
ext4: nueva generación con muchas mejoras.
Listo para producción. Son los sistemas de archivos predeterminados para todos los sistemas GNU/Linux del mundo
- ▶ El núcleo de Linux admite muchos otros sistemas de archivos: reiserFS, JFS, XFS, etc. Cada uno de ellos tiene sus propias características, pero están más orientados hacia el servidor o las cargas de trabajo científicas
- ▶ Btrfs (“Butter FS”)
La próxima generación. Gran rendimiento. Disponible en la línea principal, pero aún experimental

Recomendamos ext2 para particiones muy pequeñas (< 5 MB), porque otros sistemas de archivos necesitan demasiado espacio para metadatos (ext3 y ext4 necesitan aproximadamente 1 MB para una partición de 4 MB)



Crear volúmenes ext2/ext3/ext4

- ▶ Para crear un sistema de archivos ext2/ext3/ext4 vacío en un dispositivo de bloque o dentro de un archivo de imagen ya existente
 - ▶ `mkfs.ext2 /dev/hda3`
 - ▶ `mkfs.ext3 /dev/sda2`
 - ▶ `mkfs.ext4 /dev/sda3`
 - ▶ `mkfs.ext2 disk.img`
- ▶ Para crear una imagen del sistema de archivos desde un directorio que contiene todos sus archivos y directorios
 - ▶ Use la herramienta `genext2fs`, del paquete del mismo nombre
 - ▶ `genext2fs -d rootfs/ rootfs.img`
 - ▶ Su imagen está lista para ser transferida a su dispositivo de bloques



- ▶ Una vez que se ha creado una imagen del sistema de archivos, se puede acceder y modificar su contenido desde la estación de trabajo de desarrollo, utilizando el mecanismo **loop**

- ▶ Ejemplo:

```
genext2fs -d rootfs/ rootfs.img  
mkdir /tmp/tst  
mount -t ext2 -o loop rootfs.img /tmp/tst
```

- ▶ En el directorio `/tmp/tst`, se puede acceder y modificar el contenido del archivo `rootfs.img`
- ▶ Esto es posible gracias a `loop`, que es un controlador de núcleo que emula un dispositivo de bloques con el contenido de un archivo
- ▶ ¡No olvide ejecutar `umount` antes de usar la imagen del sistema de archivos!



<http://en.wikipedia.org/wiki/F2FS>

- ▶ Sistema de archivos optimizado para dispositivos de bloque basado en flash NAND
- ▶ Disponible en el núcleo principal de Linux
- ▶ Puntos de referencia: mejor desempeño en dispositivos flash la mayor parte del tiempo:
See <http://lwn.net/Articles/520003/>
- ▶ Detalles técnicos: <http://lwn.net/Articles/518988/>

Squashfs: <http://squashfs.sourceforge.net>

- ▶ Sistema de archivos comprimido de solo lectura para dispositivos de bloques. Se utiliza en partes de un sistema de archivos que pueden ser de solo lectura (kernel, binarios ...)
- ▶ Gran tasa de compresión y rendimiento de acceso de lectura
- ▶ Usado en la mayoría de los CD en vivo y distribuciones USB en vivo
- ▶ Admite compresión LZO para un mejor rendimiento en sistemas embebidos con CPU lentas (a expensas de una tasa de compresión ligeramente degradada)
- ▶ Ahora es compatible con el algoritmo XZ, para una tasa de compresión mucho mejor, a expensas de un mayor uso y tiempo de CPU

Puntos de referencia: (aproximadamente 3 veces más pequeño que ext3, y 2-4 veces más rápido)

http://elinux.org/Squash_Fs_Comparisons



- ▶ Necesita instalar el paquete `squashfs-tools`
- ▶ Creación de la imagen
 - ▶ En su estación de trabajo, cree su imagen del sistema de archivos:
`mksquashfs rootfs/ rootfs.sqfs`
 - ▶ Precaución: si la imagen ya existe, elimínela primero,
o use la opción `-noappend`
- ▶ Instalación de la imagen
 - ▶ Supongamos que su partición en el destino está en `/dev/sdc1`
 - ▶ Copie la imagen del sistema de archivos en el dispositivo
`dd if=rootfs.sqfs of=/dev/sdc1`
¡Tenga cuidado al usar `dd` para no sobrescribir la partición incorrecta!
- ▶ Monta tu sistema de archivos:
`mount -t squashfs /dev/sdc1 /mnt/root`



¡No es un sistema de archivos en bloque, por supuesto!
Perfecto para almacenar datos temporales en RAM: archivos de registro del sistema, datos de conexión, archivos temporales...

▶ tmpfs configuración:

Sistemas de archivos -> Pseudo sistemas de archivos

Vive en la caché de archivos de Linux. No desperdicia RAM: a diferencia de los discos RAM, no es necesario copiar archivos en la memoria caché, crece y se reduce para acomodar los archivos almacenados. Ahorra RAM: puede intercambiar páginas al disco cuando sea necesario.

▶ Cómo usar: elija un nombre para distinguir las diversas instancias de tmpfs que podría tener. Ejemplos:

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

```
mount -t tmpfs udev /dev
```

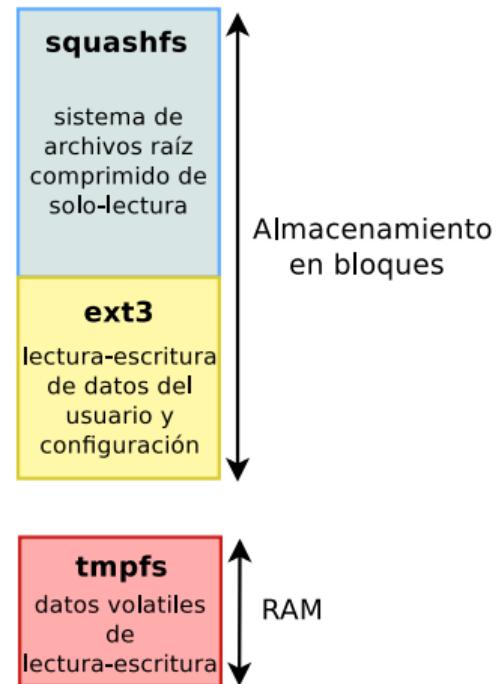
Consulte filesystems/tmpfs.txt en las fuentes del núcleo.



Mezcla de sistemas de archivos de solo lectura y lectura-escritura

Es una buena idea dividir su almacenamiento de bloques en:

- ▶ Una partición comprimida de solo lectura (`Squashfs`)
Normalmente se usa para el sistema de archivos raíz
(binarios, kernel ...).
La compresión ahorra espacio. El acceso de solo lectura
protege su sistema de errores y corrupción de datos.
- ▶ Una partición de lectura y escritura con un sistema de
archivos con registro (como `ext3`)
Se utiliza para almacenar datos de usuario o de
configuración.
Garantiza la integridad del sistema de archivos después
de apagarse o fallar.
- ▶ Almacenamiento RAM para archivos temporales
(`tmpfs`)





- ▶ Cree particiones en su almacenamiento de bloque
- ▶ Arranque su sistema con una combinación de sistemas de archivos: SquashFS para el sistema de archivos raíz (incluidas las aplicaciones), ext3 para configuración y datos de usuario, y tmpfs para archivos temporales del sistema.



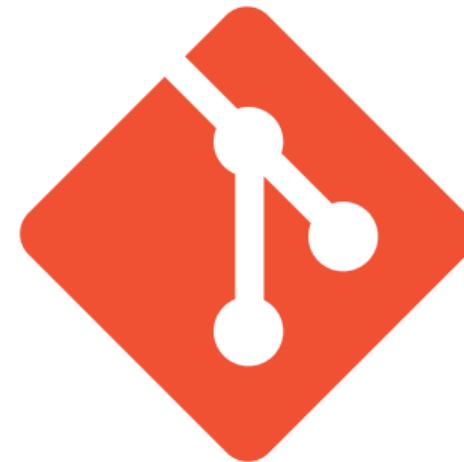
References

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





- ▶ **Embedded Linux Primer, Second Edition, Prentice Hall**

By Christopher Hallinan, October 2010

Covers a very wide range of interesting topics.

<http://j.mp/17NYxBP>

- ▶ **Building Embedded Linux Systems, O'Reilly**

By Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, Philippe Gerum
and others (including Michael Opdenacker), August 2008

<http://oreilly.com/catalog/9780596529680/>

- ▶ **Embedded Linux System Design and Development**

P. Raghavan, A. Lad, S. Neelakandan, Auerbach, Dec. 2005. Very
good coverage of the POSIX programming API (still up to date).

<http://j.mp/19X8iu2>



- ▶ **ELinux.org**, <http://elinux.org>, a Wiki entirely dedicated to embedded Linux. Lots of topics covered: real-time, filesystem, multimedia, tools, hardware platforms, etc. Interesting to explore to discover new things.
- ▶ **LWN**, <http://lwn.net>, very interesting news site about Linux in general, and specifically about the kernel. Weekly newsletter, available for free after one week for non-paying visitors.
- ▶ **Linux Gizmos**, <http://linuxgizmos.com>, a news site about embedded Linux, mainly oriented on hardware platforms related news.



Useful conferences featuring embedded Linux and kernel topics

- ▶ Embedded Linux Conference: <http://embeddedlinuxconference.com/>
Organized by the Linux Foundation: California (San Francisco, Spring), in Europe (Fall). Very interesting kernel and user space topics for embedded systems developers. Presentation slides freely available
- ▶ Linux Plumbers, <http://linuxplumbersconf.org>
Conference on the low-level plumbing of Linux: kernel, audio, power management, device management, multimedia, etc.
- ▶ FOSDEM: <http://fosdem.org> (Brussels, February)
For developers. Presentations about system development.
- ▶ Don't miss our free conference videos on
<http://free-electrons.com/community/videos/conferences/>!



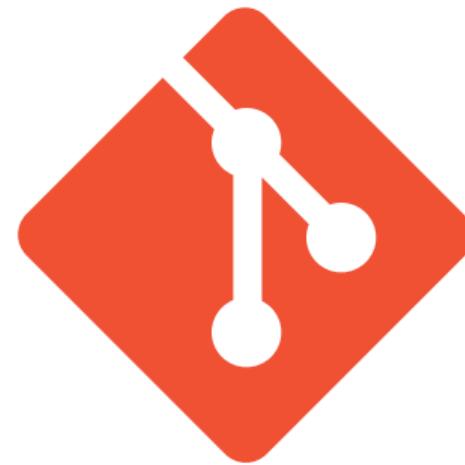
Introducción a Android

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Ecosistema de Android



¿Qué es Android?

- ▶ Sistema operativo móvil basado en nucleo de Linux
- ▶ Interfaz de usuario para pantallas táctiles
- ▶ Se utiliza en más del 80% de todos los teléfonos inteligentes
- ▶ Potencia dispositivos tales como relojes, televisores y automóviles
- ▶ Más de 2 millones de aplicaciones de Android en la tienda Google Play
- ▶ Altamente personalizable para dispositivos / por proveedores
- ▶ Código abierto



- ▶ Gestos táctiles: deslizar, tocar, pellizcar
- ▶ Teclado virtual para caracteres, números y emoji
- ▶ Soporte para Bluetooth, controladores USB y periféricos



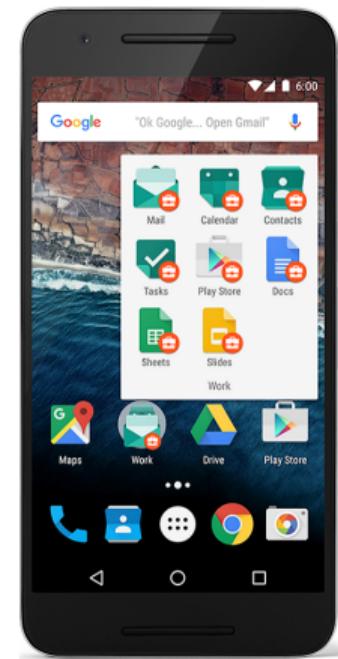
Los sensores pueden descubrir la acción del usuario y responder

- ▶ El contenido del dispositivo gira según sea necesario
- ▶ Caminar ajusta la posición en el mapa
- ▶ La inclinación conduce un automóvil virtual o controla un juguete físico
- ▶ Moverse demasiado rápido desactiva las interacciones del juego



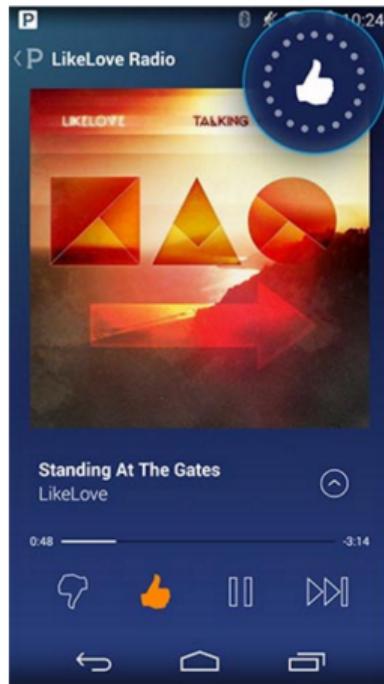
Pantalla de inicio de Android

- ▶ Iconos de inicio para aplicaciones
- ▶ Widgets auto actualizables para contenido en vivo
- ▶ Pueden ser multiples paginas
- ▶ Carpetas para organizar aplicaciones
- ▶ "OK Google"

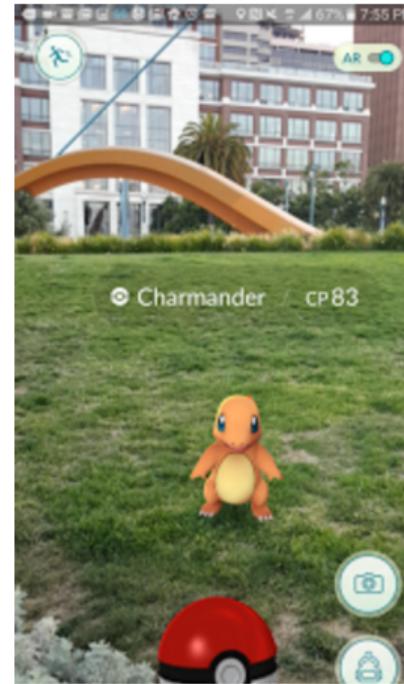




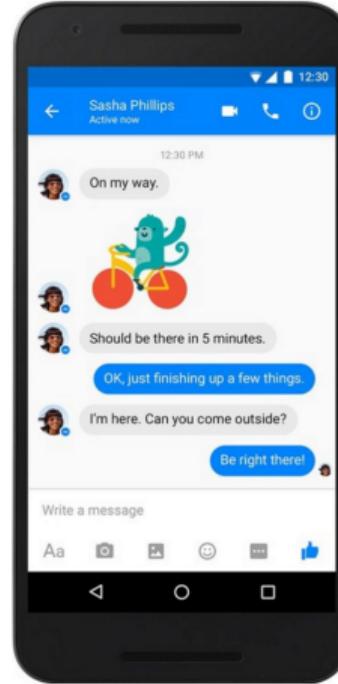
Ejemplos de aplicaciones de Android



Pandora



Pokemon GO

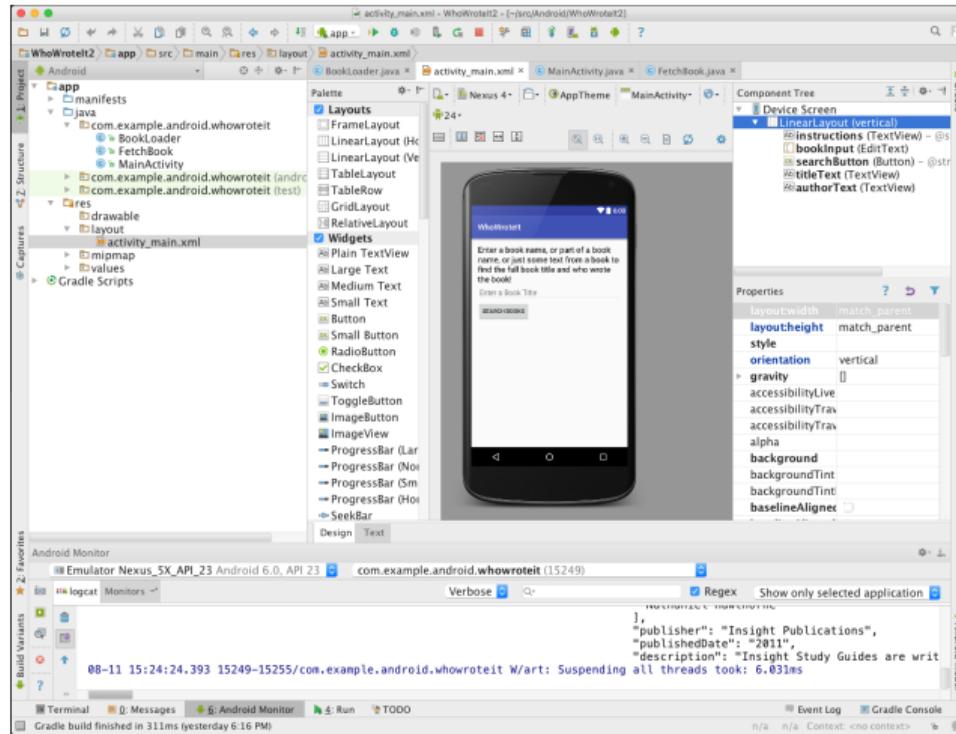
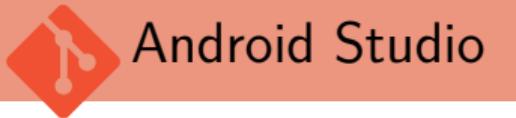


Facebook Messenger



Kit de desarrollo de software para Android (SDK)

- ▶ Herramientas de desarrollo (depurador, monitores, editores)
- ▶ Bibliotecas (mapas, wearables)
- ▶ Dispositivos virtuales (emuladores)
- ▶ Documentación (developers.android.com)
- ▶ Código de muestra



- ▶ IDE Oficial de Android
- ▶ Desarrollar, ejecutar, depurar, probar, y paquetes de aplicaciones
- ▶ Monitores y herramientas de rendimiento
- ▶ Dispositivos virtuales
- ▶ Vistas del proyecto
- ▶ Editor de diseño visual



Publicar aplicaciones a través de la tienda de Google Play:

- ▶ App Store oficial para Android
- ▶ Servicio de distribución digital operado por Google

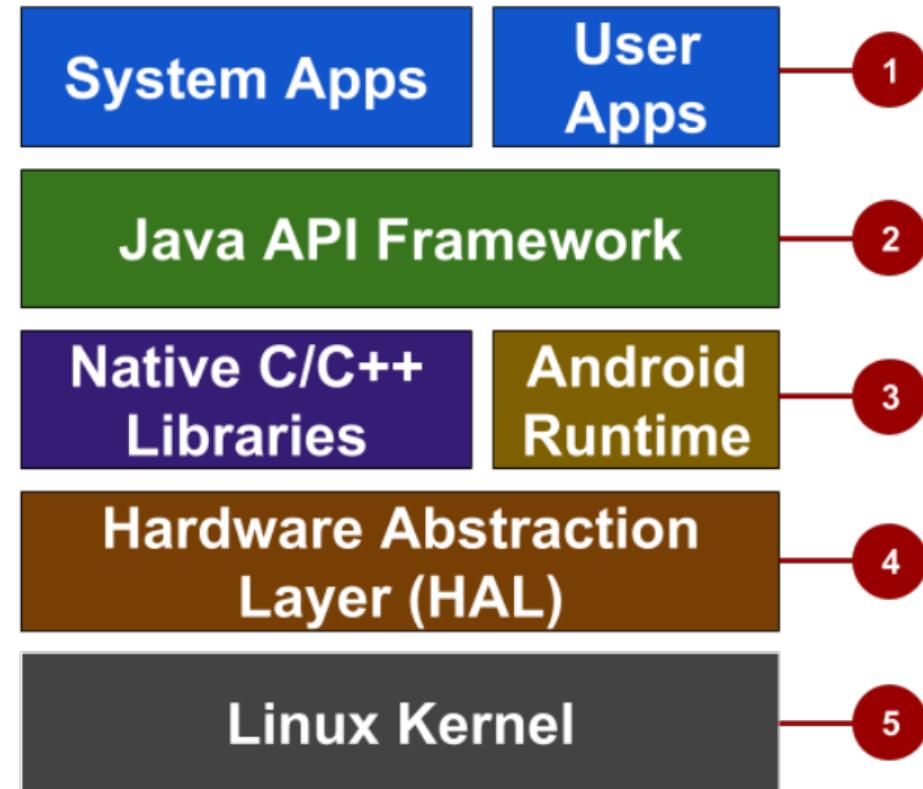




Arquitectura de la plataforma Android



- 1 Aplicaciones de sistema y usuario
- 2 API del sistema operativo Android en el marco trabajo de Java
- 3 Expone las API nativas; ejecutar aplicaciones
- 4 Expone las capacidades de hardware del dispositivo
- 5 Nucleo de Linux





Aplicaciones de sistema y usuario

- ▶ Las aplicaciones del sistema no tienen un estado especial
- ▶ Las aplicaciones del sistema proporcionan capacidades clave para los desarrolladores de aplicaciones
- ▶ Ejemplo: su aplicación puede usar una aplicación del sistema para enviar un mensaje SMS





El conjunto completo de características del sistema operativo Android está disponible para usted a través de API escritas en el lenguaje Java

- ▶ Ver jerarquía de clases para crear pantallas de interfaz de usuario
- ▶ Gestor de notificaciones
- ▶ Gestor de actividades para ciclos de vida y navegación
- ▶ Proveedores de contenido para acceder a datos de otras aplicaciones



Android en tiempo de ejecución

Cada aplicación se ejecuta en su propio proceso con su propia instancia del Android Runtime



- ▶ Las bibliotecas centrales de C/C++ brindan acceso a los componentes y servicios nativos del sistema Android



Capa de abstracción de hardware (HAL)

- ▶ Interfaces estándar que exponen las capacidades de hardware del dispositivo como bibliotecas
- ▶ Ejemplos: cámara, módulo bluetooth



- ▶ Subprocesos y gestión de memoria de bajo nivel
- ▶ Características de seguridad
- ▶ Controladores



Android History and Platform Versions for more and earlier versions before 2011



Desarrollo de aplicaciones



¿Qué es una aplicación de Android?

Una o más pantallas interactivas

- ▶ Escrito usando lenguaje de programación Java y XML
- ▶ Utiliza el Kit de desarrollo de software de Android (SDK)
- ▶ Utiliza bibliotecas de Android y el marco de aplicación de Android
- ▶ Ejecutado por la máquina virtual de Android Runtime (ART)



- ▶ Múltiples tamaños de pantalla y resoluciones
- ▶ Rendimiento: haga que sus aplicaciones sean responsivas y fluidas
- ▶ Seguridad: mantener seguros el código fuente y los datos del usuario
- ▶ Compatibilidad: corre bien en versiones anteriores de la plataforma
- ▶ Marketing: entender el mercado y sus usuarios
- ▶ (Pista: no tiene que ser costoso, pero puede serlo)



- ▶ Recursos: diseños, imágenes, cadenas, colores como XML y archivos multimedia
- ▶ Componentes: actividades, servicios, ..., y clases auxiliares como código Java
- ▶ Manifiesto: información sobre la aplicación para el tiempo de ejecución
- ▶ Configuración de la construcción: versiones APK en los archivos de configuración de Gradle



- ▶ Una **actividad** es una sola pantalla con una interfaz de usuario
- ▶ Un **servicio** realiza tareas de larga duración en segundo plano
- ▶ Un **proveedor de contenido** gestiona conjunto de datos compartidos
- ▶ Un **receptor de difusión** responde a los anuncios de todo el sistema



Piense en Android como un hotel

- ▶ Tu aplicación es el invitado
- ▶ El sistema Android es el gerente del hotel
- ▶ Los servicios están disponibles cuando los solicite (intenciones)
 - ▶ En primer plano (actividades) como el registro
 - ▶ En el fondo (servicios) como lavandería
- ▶ Te llama cuando un paquete ha llegado (receptor de difusión)
- ▶ Acceder a las empresas turísticas de la ciudad (proveedor de contenidos).



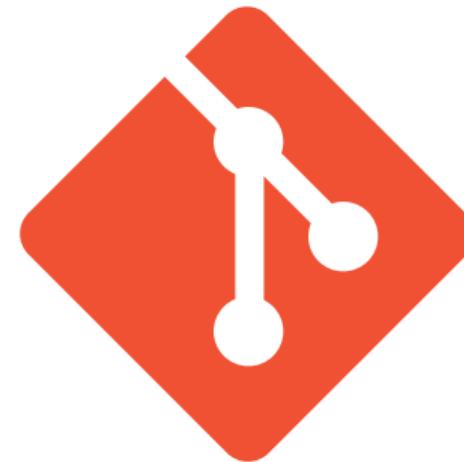
Vistas, Diseño y Recursos

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Vistas



Todo lo que ves es una vista

Si observa su dispositivo móvil, cada elemento de la interfaz de usuario que ve es una Vista.





¿Qué es una vista?

Las vistas son los bloques de construcción básicos de la interfaz de usuario de Android

- ▶ mostrar texto (clase TextView), editar texto (clase EditText)
- ▶ Botones (clase de botones), menús, otros controles
- ▶ desplazable (ScrollView, RecyclerView)
- ▶ mostrar imágenes (ImageView)
- ▶ subclase de clase View

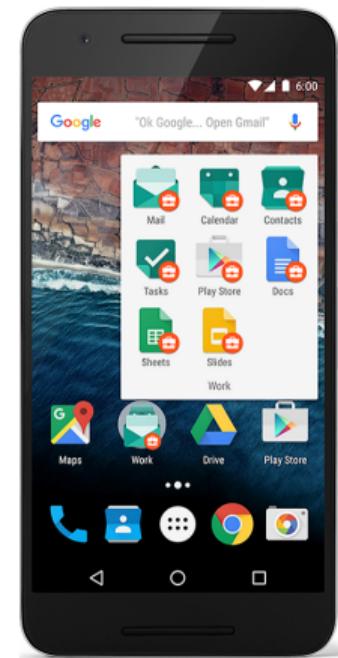


Las vistas tienen propiedades

- ▶ Tener propiedades (por ejemplo, color, dimensiones, posicionamiento)
- ▶ Puede tener foque (por ejemplo, seleccionado para recibir la entrada del usuario)
- ▶ Puede ser interactivo (responder a los clicks de los usuarios)
- ▶ Puede ser visible o no
- ▶ Tener relaciones con otras vistas



Ejemplos de vistas

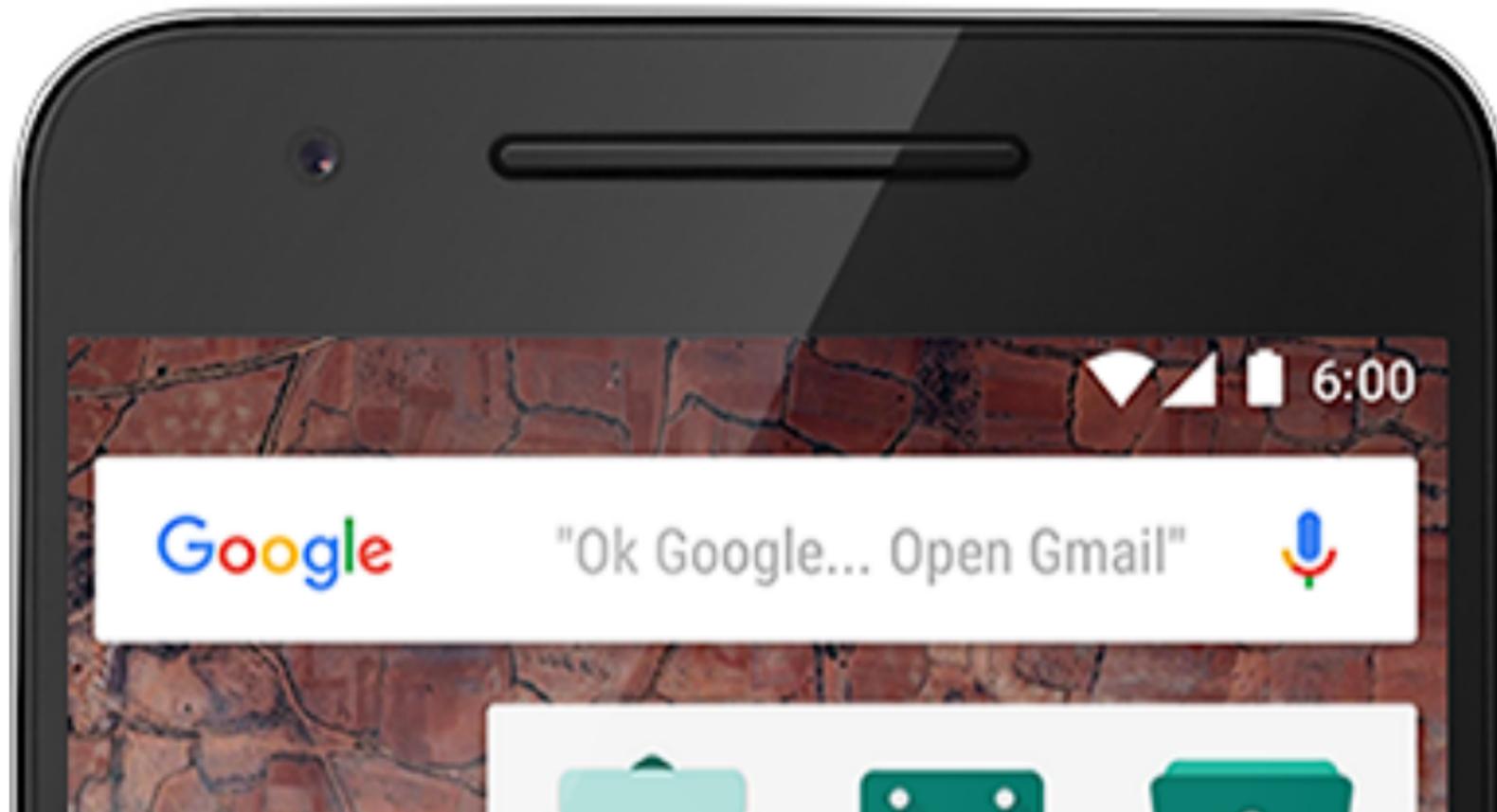




- ▶ Gráficamente dentro de Android Studio
- ▶ Editor XML
- ▶ De forma codificada en Java



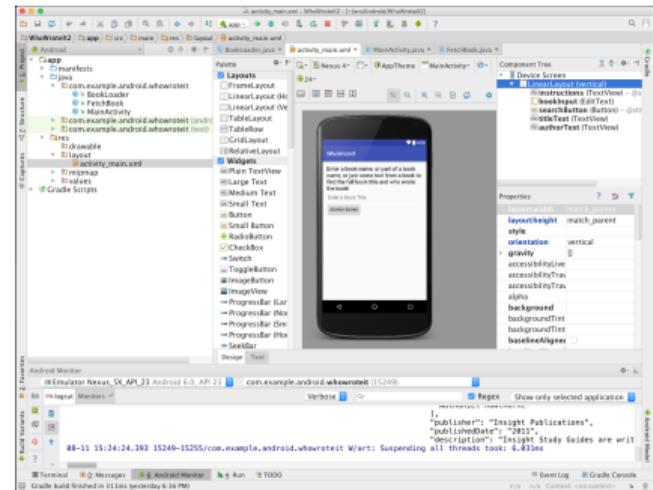
Vistas definidas en el editor de diseño





Usando el editor de diseño

- ▶ Gestor de redimensión
- ▶ Gestor de línea de restricción
- ▶ Gestor de línea base
- ▶ Gestor de restricción





Vistas definidas en XML

```
<TextView  
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"  
/>
```



Tareas en segundo plano

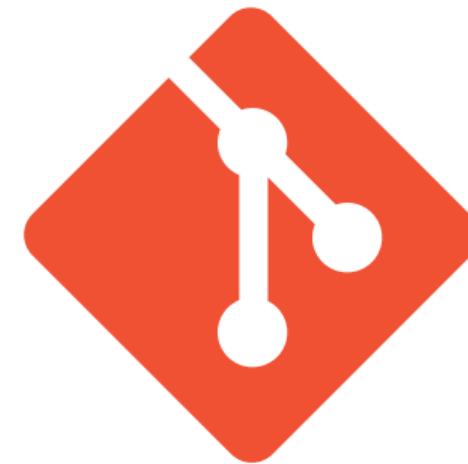
Tareas en segundo plano

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Hilos



El hilo principal

- ▶ Independent path of execution in a running program
- ▶ Code is executed line by line
- ▶ App runs on Java thread called "main" or "UI thread"
- ▶ Draws UI on the screen
- ▶ Responds to user actions by handling UI events



El hilo principal debe ser veloz

- ▶ Hardware updates screen every 16 milliseconds
- ▶ UI thread has 16 ms to do all its work
- ▶ If it takes too long, app stutters or hangs



Users uninstall unresponsive

- ▶ If the UI waits too long for an operation to finish, it becomes unresponsive
- ▶ The framework shows an Application Not Responding (ANR) dialog



What is a long running task?

- ▶ Network operations
- ▶ Long calculations
- ▶ Downloading/uploading files
- ▶ Processing images
- ▶ Loading data



Background threads

- ▶ Execute long running tasks on a background thread
- ▶ AsyncTask
- ▶ The Loader Framework
- ▶ Services



Two rules for Android threads



Guardando el estado de una Actividad 1/2

- ▶ Como las aplicaciones tienden a ser eliminadas y reiniciadas muy seguido, necesitamos una manera de almacenar el estado interno cuando se eliminan y recargan al reiniciarse
- ▶ Una vez más, esto se realiza a través de los callbacks
- ▶ Antes de eliminar la aplicación, el sistema llama al método callback `onSaveInstanceState` y cuando reinicia llama a `onRestoreInstanceState`
- ▶ En ambos casos, provee un Bundle como argumento para permitir a la actividad almacenar lo que es necesario y lo recarga más tarde, con una pequeña sobrecarga overhead



Guardando el estado de una actividad 2/2

- ▶ Esto hace que la creación / supresión de las actividades sea segura para el usuario, mientras que permite ahorrar la cantidad de memoria requerida
- ▶ Estas callbacks no se invocan siempre. Si la actividad es eliminada porque el usuario la dejó de forma permanente (a través del botón back), no se va a invocar
- ▶ Por defecto, estas actividades también se invocan cuando se rota el dispositivo, dado que la actividad será eliminada y reinicializada por el sistema para cargar nuevos recursos



Ciclo de vida de las Actividades



Callbacks de las Actividades



Actividad HelloWorld 1/2

```
public class ExampleActivity extends Activity {  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.example);  
        Log.i("ExampleActivity", "Activity created!");  
    }  
  
    protected void onStart() {  
        super.onStart();  
    }  
  
    protected void onResume() {  
        super.onResume();  
    }  
}
```



Actividades

- ▶ Las Actividades representan una única pantalla de la interfaz de usuario de la aplicación
- ▶ Se ensamblan para proveer una interfaz consistente. Si tomamos como ejemplo una aplicación de email, tenemos:
 - ▶ Una actividad escuchando los emails recibidos
 - ▶ Una actividad para redactar los nuevos emails
 - ▶ Una actividad para leer los nuevos emails
- ▶ Otras aplicaciones pueden necesitar de estas actividades. Para continuar con el ejemplo, la aplicación de la Cámara puede querer iniciar la actividad para poder compartir una imagen
- ▶ Es responsabilidad del desarrollador de la aplicación informar de las actividades disponibles al sistema
- ▶ Cuando una actividad inicia una nueva actividad, la última reemplaza a la inicial en la pantalla y la misma se ubica en la pila de (*back stack*) la cual mantiene las últimas actividades usadas, para que, cuando el usuario haya terminado con la nueva actividad, puede de forma simple volver a la anterior



Back Stack

Creditos: <http://developer.android.com>



Back Stack

Creditos: <http://developer.android.com>



- ▶ Dado que no hay un punto de entrada único y el sistema administra las actividades, las mismas deben definir callbacks que el sistema puede llamar en algún momento en el tiempo
- ▶ Las actividades pueden estar en alguno de estos estados, dependiendo de como interactúa con el usuario

En ejecución La actividad está en primer plano y tiene foco

Pausado La actividad todavía es visible en la pantalla, pero no tiene más el foco. Puede ser destruida por el sistema ante un requerimiento grande de memoria

Detenido La actividad no es más visible en la pantalla. Puede ser destruida en cualquier momento por el sistema

Destruido La actividad fue detenida por el sistema llamando a su método `finish()`



- ▶ Hay callbacks para cada cambio desde uno de los estados hacia otro
- ▶ Los más importantes son `onCreate` y `onPause`
- ▶ Todos los componentes de una aplicación se ejecutan en el mismo hilo. Si se realizan operaciones largas en los callbacks, se va a bloquear la aplicación entera (IU incluida). Siempre debe utilizar hilos para las tareas de larga duración.



Ciclo de vida de las Actividades 3/3

Creditos: <http://developer.android.com>



Guardando el estado de una Actividad 1/2

- ▶ Como las aplicaciones tienden a ser eliminadas y reiniciadas muy seguido, necesitamos una manera de almacenar el estado interno cuando se eliminan y recargan al reiniciarse
- ▶ Una vez más, esto se realiza a través de los callbacks
- ▶ Antes de eliminar la aplicación, el sistema llama al método callback `onSaveInstanceState` y cuando reinicia llama a `onRestoreInstanceState`
- ▶ En ambos casos, provee un Bundle como argumento para permitir a la actividad almacenar lo que es necesario y lo recarga más tarde, con una pequeña sobrecarga overhead



- ▶ Esto hace que la creación / supresión de las actividades sea segura para el usuario, mientras que permite ahorrar la cantidad de memoria requerida
- ▶ Estas callbacks no se invocan siempre. Si la actividad es eliminada porque el usuario la dejó de forma permanente (a través del botón back), no se va a invocar
- ▶ Por defecto, estas actividades también se invocan cuando se rota el dispositivo, dado que la actividad será eliminada y reinicializada por el sistema para cargar nuevos recursos



Ciclo de vida de las Actividades

Creditos: <http://developer.android.com>



Callbacks de las Actividades

Creditos: <http://developer.android.com>



Actividad HelloWorld 1/2

```
public class ExampleActivity extends Activity {  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.example);  
        Log.i("ExampleActivity", "Activity created!");  
    }  
  
    protected void onStart() {  
        super.onStart();  
    }  
  
    protected void onResume() {  
        super.onResume();  
    }  
}
```



- ▶ Escriba una aplicación que utilice hilos para realizar una tarea de fondo
- ▶ Escriba una aplicación que utilice AsyncTask para realizar una tarea de fondo



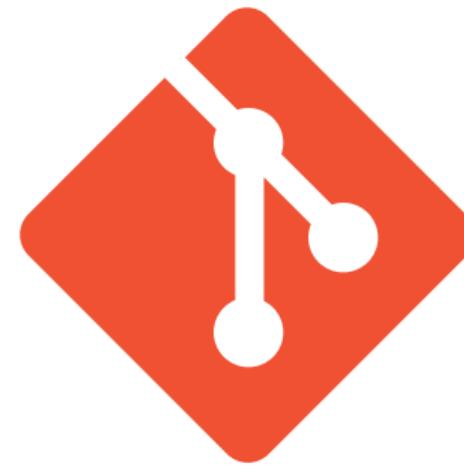
Last slides

Free Electrons

© Copyright 2022, Luciano Diamand.

Creative Commons BY-SA 3.0 license.

Correcciones, sugerencias, contribuciones y traducciones son bienvenidas!





Thank you!
And may the Source be with you