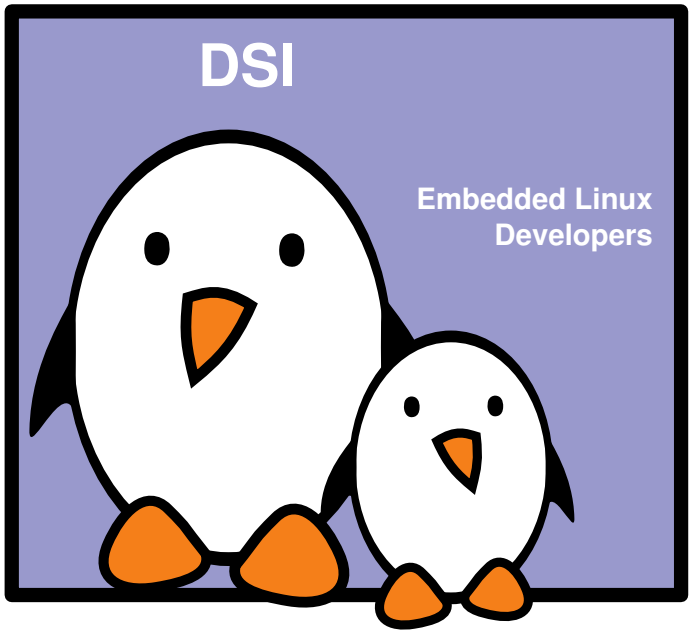
Gestor de arranque



Authors

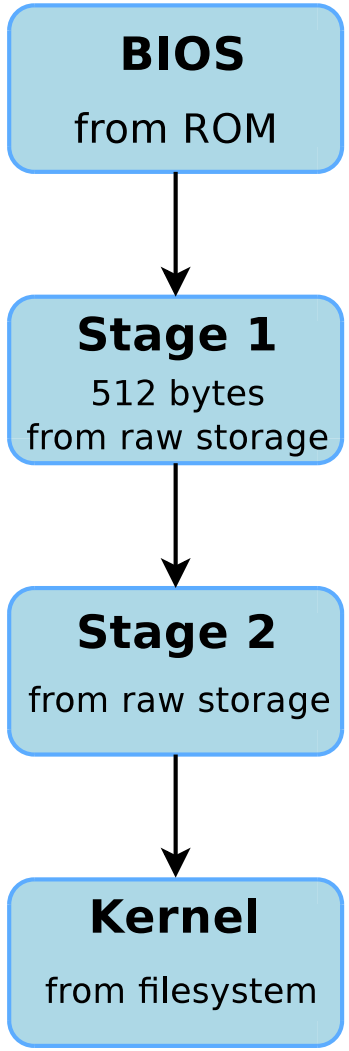
Secuencia de inicio

Gestores de arranque

- El gestor de arranque es una pieza 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 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



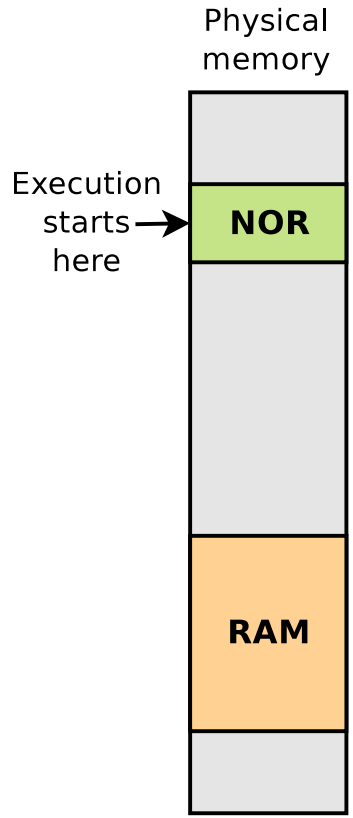
Gestor de arranque en x86 2/2

- 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

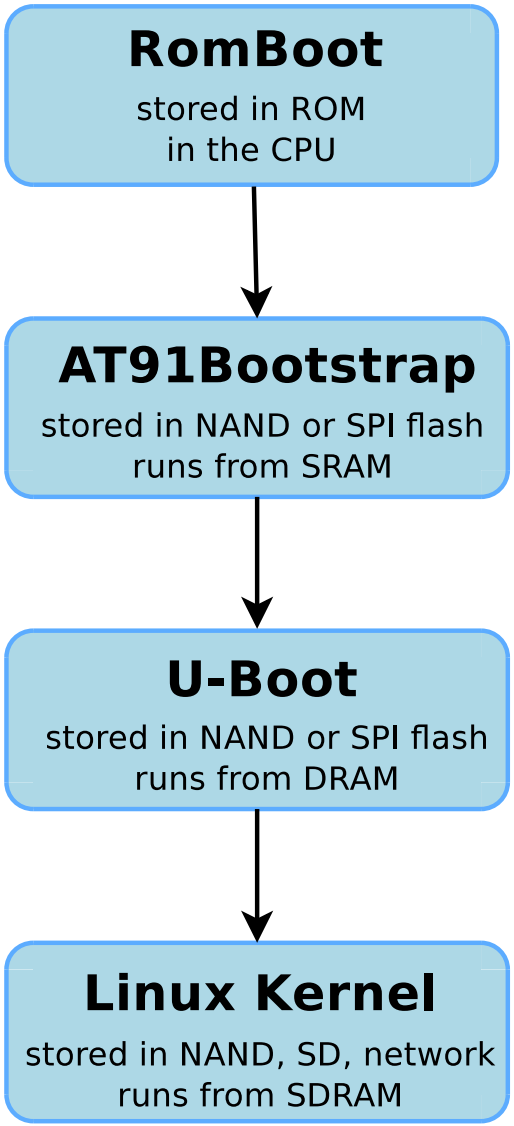
- Cuando se activa, la CPU comienza a ejecutar código en una dirección fija
- No hay ningún otro mecanismo de arranque proporcionado por la CPU
- El diseño del hardware debe garantizar que un chip flash NOR esté conectado para que sea accesible en la dirección en la que la CPU comienza a ejecutar instrucciones
- 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

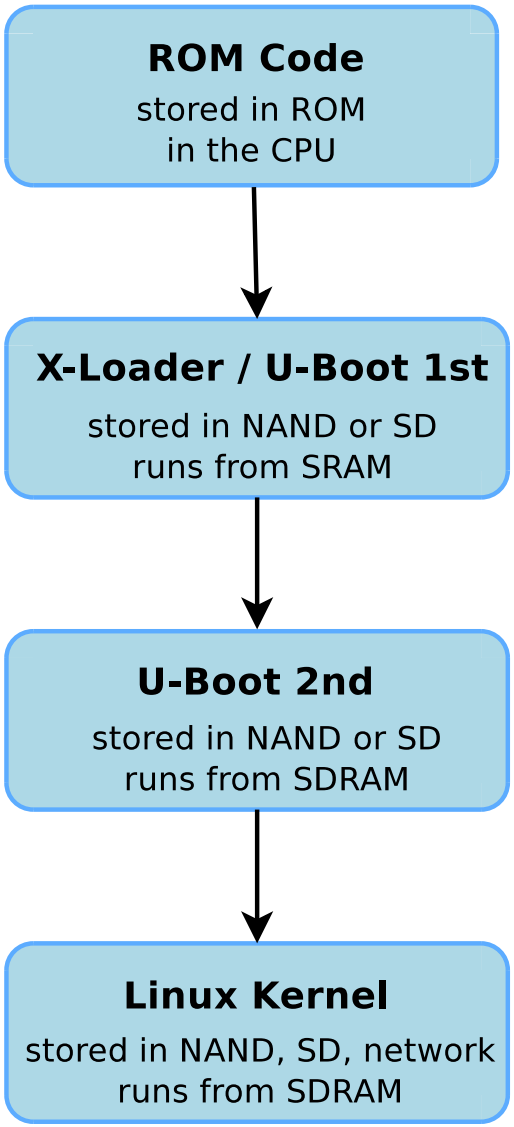
- La CPU tiene un código de arranque integrado en ROM
  - BootROM en CPUs AT91, "código ROM" en OMAP, etc.
  - Los detalles exactos dependen de la CPU
- Este código de arranque es capaz de cargar un gestor de arranque de primer nivel desde un almacenamiento en una SRAM interna (la DRAM no se ha inicializado 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



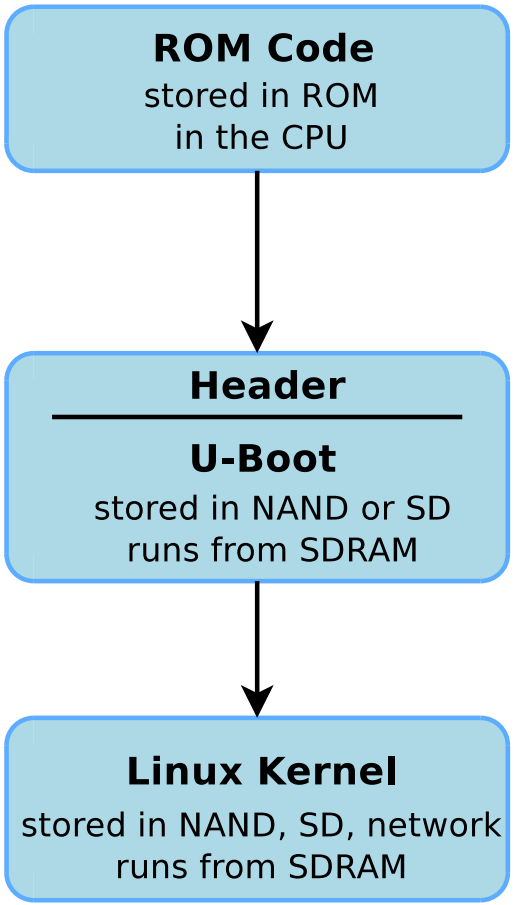
- **RomBoot**: intenta encontrar una imagen de arranque válida desde varias fuentes de almacenamiento y cargarlo en SRAM (DRAM no inicializada todavía). Tamaño limitado a 4 KB. Ninguna interacción del usuario es posible en el modo de arranque estándar.
- **AT91Bootstrap**: se ejecuta desde SRAM. Inicializa 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 OMAP3



- **ROM Code**: intenta encontrar una imagen de arranque válida desde varias fuentes de almacenamiento y cargarlo en SRAM o RAM (la RAM puede ser inicializada por el código ROM a través de un encabezado de configuración). Tamaño limitado a <64 KB. No hay interacción del usuario posible.
- **X-Loader** o **U-Boot**: 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. Recupera el sistema completamente (el gestor no está más disponible).

Arranque en Marvell SoC



- **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

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 versions se llamana `YYYY.MM`.

Configuración de U-Boot

- 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
[...]
```

Configuración y compilación de U-Boot

- 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.

Instalando U-Boot

- 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 version
  - U-Boot ya está instalado y puede utilizarse para grabar una nueva versión de U-Boot. Sin embargo, tenga cuidado: i 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 ejecutandose 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 #255000029ff8000001685802120290l1
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 (RO) Bootstrap
Area 1: C0002000 to C0003FFF Environment
Area 2: C0004000 to C0041FFF (RO) 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)

- 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)

- `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 propositos de prueba.

Comandos de variables de entorno (1)

- 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 de variables de entorno (2)

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

Scripts en variables de entorno

- 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'`

Transfiriendo archivos hacia el destino

- 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

TFTP

- 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.

Laboratorio práctico - U-Boot

Es momento de iniciar la práctica de laboratorio!

- Comunicarse 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