Bootloaders

Mg. Ing. Gonzalo E. Sanchez
MSE - 2022

Implementación de Sistemas Operativos II

Bootloaders

- Secuencia de booteo
- U-Boot como bootloader

Secuencia de booteo

Secuencia de booteo

- Un bootloader es básicamente una porción de código que efectúa las siguientes tareas:
  - Inicialización de hardware básica
  - Carga de una aplicación binaria, usualmente un kernel de OS.
  - La carga se hace desde un dispositivo de almacenamiento, red u otro tipo de memoria no volátil.
  - Puede (o no) tener posibilidad de descompresión para la aplicación binaria.
  - Ejecución de la aplicación binaria.

Secuencia de booteo

- Además de todas las prestaciones básicas mencionadas, los bootloaders suelen tener un shell para ejecutar comandos.

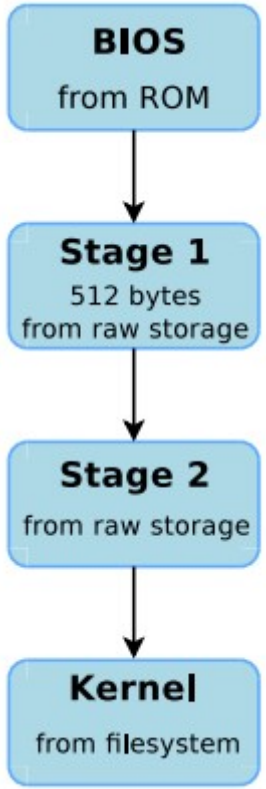
- Estos comandos pueden ser:
  - Cargar datos desde memoria o desde una red.
  - Inspección de memoria.
  - Diagnósticos y testeo de hardware.

Secuencia de booteo

- EJEMPLO: en el caso de los procesadores x86 existe una memoria no volátil vinculada: la BIOS.
- El programa contenido en la bios se ejecuta por el CPU después de un reset.
- Es responsable de la inicialización básica del hardware y la carga de una porción pequeña de código.
- Usualmente este código cargado ocupa los primeros 512 bytes de un dispositivo de almacenamiento.

Secuencia de booteo

- Este código de 512 bytes es usualmente el primer bootloader (llamado primera etapa).
- Tiene como objetivo cargar el bootloader completo.
- Al cargarse completo, el bootloader ofrece sus funcionalidades completas.
- Lo general es que tenga soporte para formatos de filesystems.
- Esto implica que se puede cargar el kernel de un filesystem normal

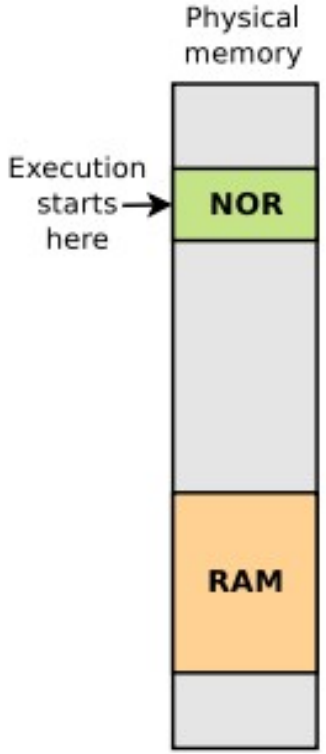


Secuencia de booteo

- Uno de los más conocidos y poderosos: GRUB (GRand Unified Bootloader).
- Puede leer muchos formatos de filesystems, permitiendo cargar el kernel de cualquiera de ellos.
- Provee un shell muy poderoso, con variedad de comandos.
- Otro muy difundido es Syslinux, utilizado para network y booting desde dispositivos extraíbles.

Secuencia de booteo

- Como ejemplo para una SBC, el CPU siempre inicia en una dirección fija definida.
- No existe otro mecanismo de booting provisto por el CPU.
- El diseño de hardware debe asegurar que existe un NOR flash chip accesible al CPU al momento del reset.



Secuencia de booteo

- La primera etapa del bootloader debe ser programada en esta dirección.
- Que el chip sea NOR es mandatorio dado que permite Acceso aleatorio (una memoria NAND no lo permite).
- Ya no es muy común porque es muy poco práctico y requiere un chip flash NOR

Secuencia de booteo

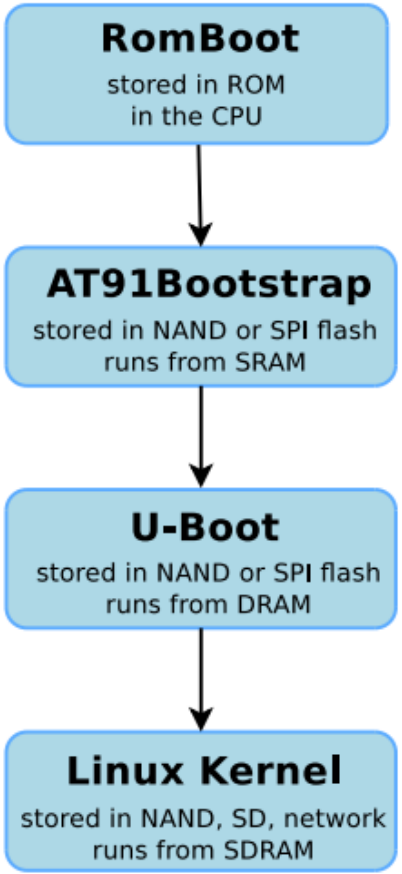
- Otro caso es cuando la CPU tiene el código de booteo integrado en ROM.
- Ejemplos: BootROM en CPUs AT91, “ROM Code” en OMAP
- Este pequeño código es capaz de cargar un bootloader stage 1 desde un dispositivo de almacenamiento.
- El dispositivo de almacenamiento es usualmente una MMC, NAND, flash SPI, etc.
- El stage 1 del bootloader es limitado en tamaño dado las limitaciones de hardware (cargado en SRAM).

Secuencia de booteo

- Aclaración: la SRAM es más rápida y usualmente más cara que la DRAM, por lo que usualmente es pequeña.
- El stage 1 del bootloader puede ser provisto por el CPU vendor o bien ser un proyecto de comunidad.
- Este primer nivel de bootloader debe inicializar la DRAM y otro hardware para cargar el segundo stage del bootloader.

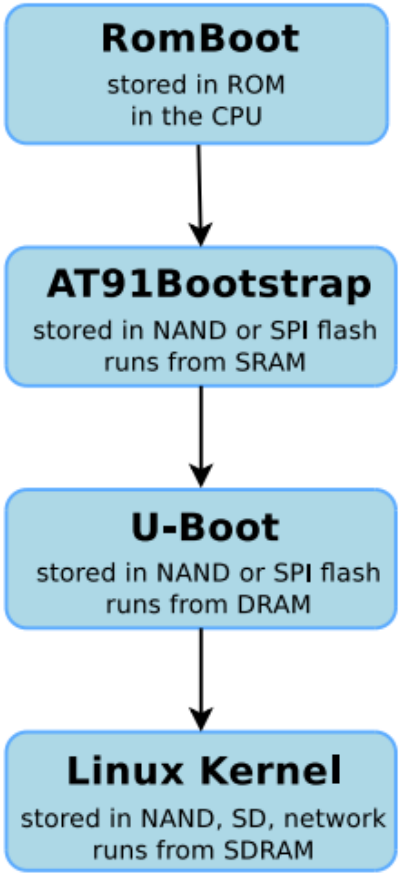
Secuencia de booteo

- **EJEMPLO**: Booteo de un ARM Atmel AT91.
- **RomBoot**: Trata de encontrar una imagen valida de bootstrap desde varios lugares.
- Una vez encontrada se carga en SRAM (DRAM todavía no inicializada).
- Limitación de tamaño: 4KB
- No es posible ninguna interacción con el usuario.



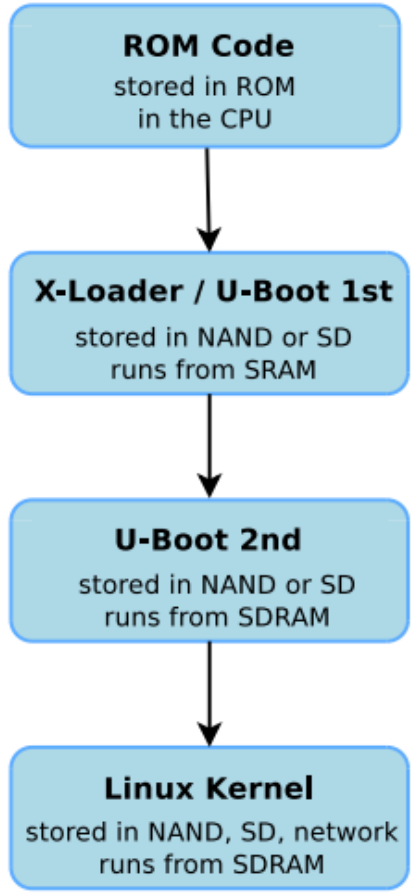
Secuencia de booteo

- **AT91Bootstrap:** corre desde SRAM.
- Inicializa la DRAM, el controlador NAND o el SPI.
- Carga el segundo stage de bootloader en RAM. No es posible interacción con usr.
- **U-Boot**. Corre desde RAM, inicializa algunos otros dispositivos de hardware
- Carga la imagen de kernel a RAM y provee ya una línea de comandos (shell).



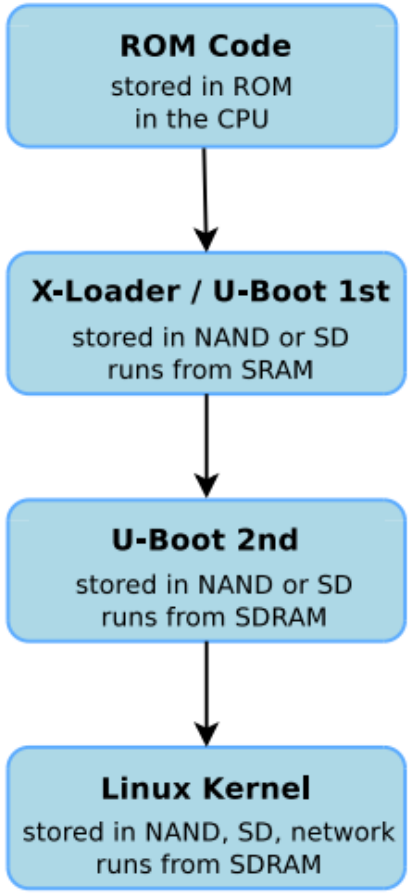
Secuencia de booteo

- **EJEMPLO**: Booteo de un ARM TI OMAP3.
- **ROM Code**: trata de encontrar una imagen bootstrap válida de varios lugares.
- Carga la imagen bootstrap en SRAM o RAM (la RAM puede ser inicializada).
- Tamaño limitado a 64KB.
- No es posible la interacción del usuario



Secuencia de booteo

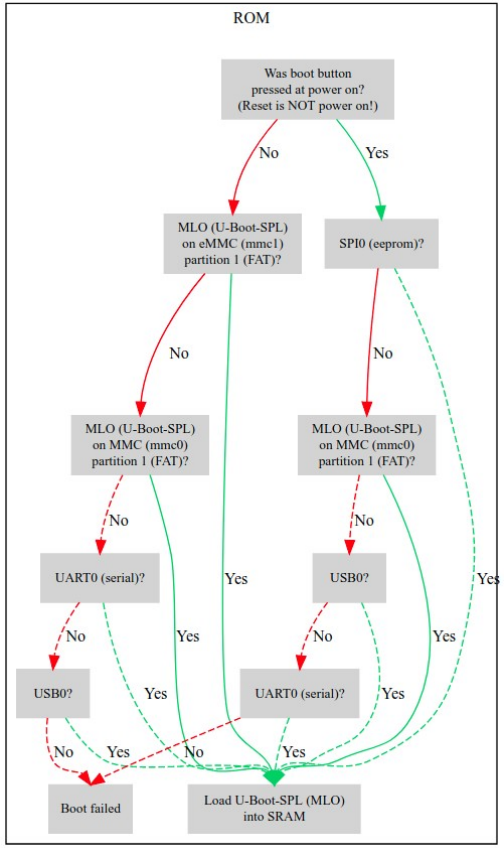
- **X-Loader o U-Boot:** Corre desde SRAM, inicializa la DRAM, MMC, NAND.
- Carga el bootloader secundario en RAM y lo inicia. No es posible interacción de usr.
- Este archivo es llamado **MLO.**
- **U-Boot:** Corre desde RAM. Inicializa otros dispositivos de hardware.
- Carga la imagen de kernel en RAM y lo inicia.



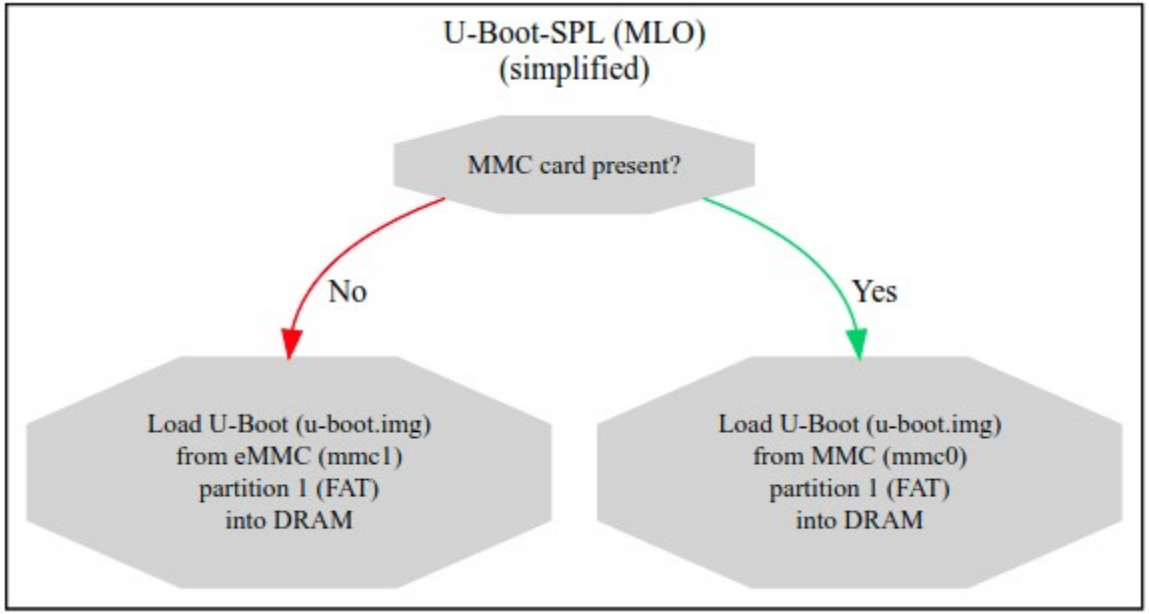
Secuencia de booteo

- Enfoque en el main bootloader, que es la parte genérica que ofrece las funcionalidades más importantes.
- Hay varios bootloaders genéricos open-source. Los más populares son:
  - **U-Boot**. El bootloader universal, más utilizado para ARM. De-facto estándar hoy en dia.
  - **Barebox**. Un bootloader de arquitectura neutral. Mejor diseño y mejor código, pero no tiene tanto soporte como U-boot.

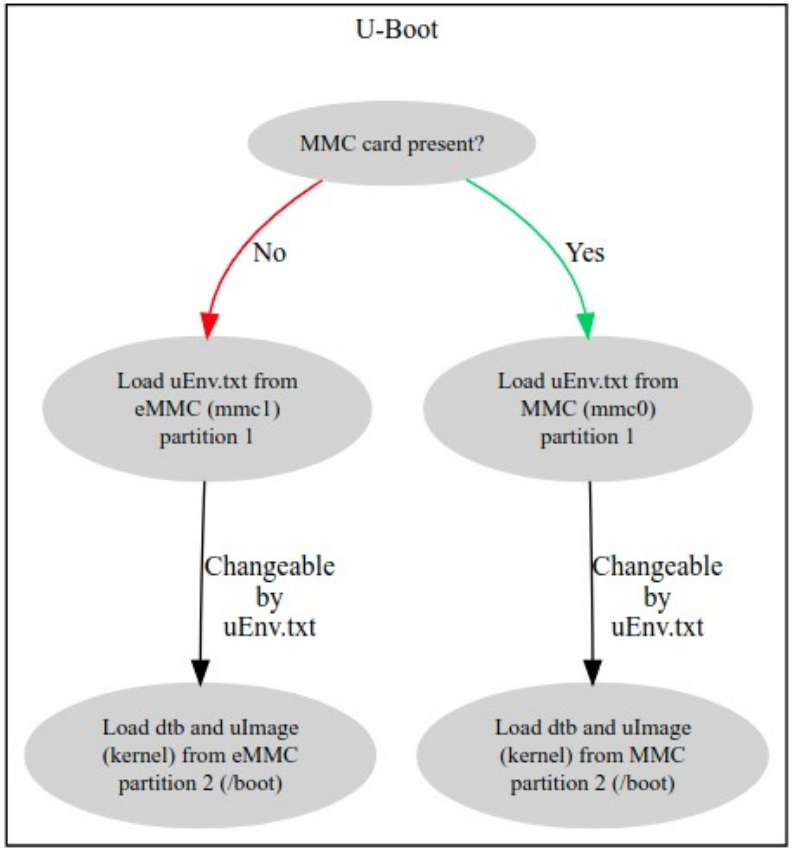
Secuencia de booteo - BeagleBone Black



Secuencia de booteo - BeagleBone Black



Secuencia de booteo - BeagleBone Black



U-Boot

U-Boot

- El proyecto U-Boot es un típico proyecto de software libre.
- Licencia GPLv2 (la misma que utiliza Linux).
- Disponible de forma gratuita aquí.
- Muy bien documentado.
- Todas las fuentes están disponibles a través de un repositorio Git.
- Desde el 2008 sigue un calendario de releases por intervalos regulares (cada tres meses).

U-Boot

- Una vez que se clona el repositorio, accedemos al directorio **include/configs/**
- En este directorio existe un archivo por cada hardware soportado.
- Estos archivos definen:
  - El tipo de CPU utilizada
  - Los periféricos y sus configuraciones.
  - Las funcionalidades que deben ser incluidas en U-boot al compilar.

U-Boot

- A fin de cuentas, los archivos mencionados son simples headers (.h) que contienen constantes de preprocesador.
- Esto puede verse en más detalle en el archivo README.
- Dentro del archivo se pueden ajustar las constantes para agregar o quitar funcionalidades.
- Asumiendo que la SBC a utilizar esta soportada por U-boot, debería haber una entrada correspondiente en **boards.cfg**

U-Boot

- Como en el caso del kernel de linux, U-boot debe ser configurado antes de ser compilado.
- Esto se hace mediante el comando **make BOARDNAME_config.**
- **BOARDNAME** es el nombre de la SBC, como se muestra en la primer columna de **boards.cfg**
- Antes de compilar, el cross-compilador debe estar disponible en la variable **PATH**.
- De la misma manera que para el kernel se debe especificar el cross-compilador (**make CROSS_COMPILE=arm-linux-**),

U-Boot

- El resultado de la compilación sera un archivo llamado **u-boot.bin**
- Este archivo es la imagen de U-Boot.
- Dependiendo de la SBC utilizada, puede haber otros archivos generados:
  - u-boot.img
  - MLO
  - u-boot.kwb
  - etc...

U-Boot

- Para utilizar U-boot generalmente es necesario instalarlo en una memoria flash, así ejecutarse por el hardware.
- Dependiendo el hardware, la instalación de U-Boot se hace de distintas maneras:
  - El CPU provee alguna clase de monitor específico de booteo, con el cual el usuario puede comunicarse mediante el puerto serie.
  - El CPU bootea primero sobre un dispositivo removible (MMC) antes de bootear desde uno fijo (NAND).
    - En este caso, es necesario bootear desde MMC para cargar una nueva versión en NAND.

U-Boot

- Dependiendo el hardware, la instalación de U-Boot se hace de distintas maneras:
  - U-Boot ya está instalado, y puede ser utilizado para grabar una nueva versión de U-Boot.
    - En este caso hay que tener sumo cuidado: si la nueva versión de U-boot no funciona, la SBC será un hermoso pisapapeles.
  - La SBC provee una interfaz JTAG que permite grabar sobre la memoria flash de manera remota, sin necesidad de correr un sistema en la SBC.
    - Esto permite rescatar la SBC si el bootloader no funciona.

U-Boot

- La consola de U-Boot ofrece un set de comandos útiles.
- Podemos destacar algunos:
  - Flash information (NOR y SPI): **flinfo**
  - NAND flash information: **nand info**
  - Version details: **version**
- El set de comandos depende de la configuración de U-boot.

U-Boot

- Otros comandos útiles:
  - **help**
  - **boot**. Corre el comando boot por defecto, guardado en **bootcmd**.
  - **bootm** <addrr>. Inicia una imagen de kernel almacenada en la dirección RAM especificada.
  - **ext2load** carga un archivo desde un fs ext2 en RAM.
  - **fatload** carga un archivo desde un fs fat en RAM.
  - **tftp** carga un archivo desde la red en RAM.
  - **ping**

U-Boot

- Otros comandos útiles:
  - **loadb, loads, loady** cargan un archivo desde la línea serie a RAM.
  - **usb**. Inicializa y controla el subsystem USB, utilizado más que nada para dispositivos de almacenamiento USB.
  - **mmc** Inicializa y controla el subsystem MMC, utilizado para memorias SD.
  - **nand** para leer, escribir y borrar contenidos en flash NAND.
  - **erase, protect, cp**, para borrar, modificar protección y escribir sobre flash NOR.

U-Boot

- Otros comandos útiles:
  - **md** muestra los contenidos de memoria. Es útil para chequear lo que está cargado en RAM.
  - **mm** modifica contenidos en memoria. Útil para modificar directamente registros de hardware para testeo.

U-Boot

- U-boot puede ser configurado mediante variables de ambiente, afectando el comportamiento de distintos comandos.
- Las variables de ambiente se cargan desde flash a RAM, en el momento que U-boot inicia.
- Esto hace que puedan ser modificadas y guardadas para persistencia entre PORs
- Hay una ubicacion dedicada en flash/MMC para guardar el environment de U-boot, definido en el archivo de configuración.

U-Boot

- Comandos para manipular las variables de ambiente (environment):
  - printenv
  - printenv <variable-name>
  - setenv <variable-name> <variable-value>
  - editenv <variable-name>
  - saveenv

U-Boot

- Variables de ambiente importantes:
  - **bootcmd**. Contiene el comando que U-Boot ejecuta automaticamente luego de un delay configurable si el proceso no se interrumpe.
  - **bootargs**. Contiene los argumentos que se pasan al kernel de linux.
  - **serverip** contiene la direccion IP para comandos relacionados con network.
  - **ipaddr** contiene el IP que utiliza U-Boot.
  - **netmask** contiene la mascara de subred.
  - **ethaddr** es la direccion MAC

U-Boot

- Variables de ambiente importantes:
  - **bootdelay** es el delay en segundos antes de que U-Boot ejecute bootcmd.
  - **autostart** cuando está seteada a “yes”, hace que U-Boot inicie automáticamente una imagen que se cargó en memoria.
- Las variables de ambiente pueden contener pequeños scripts para ejecutar varios comandos y testear el resultado de ellos.
- Esto es muy útil para automatizar en proceso de booteo o de upgrade.

U-Boot

- Se pueden hacer test utilizando:
  if command ; then ... ; else ... ; fi
- Los scripts deben correrse mediante **run <variable-name>.**
- Se puede hacer referencia a otras variables utilizando **${var-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

- U-Boot es mayormente utilizado para cargar y bootear una imagen de kernel
- También permite cambiar la imagen de kernel y el root filesystem cargado en flash.
- Los archivos deben necesariamente intercambiarse entre el target y el host, esto es posible mediante:
  - Network (muy practico y rapido).
  - Memoria SD (mucho más lento y tedioso).
  - Puerto Serie (extremadamente lento).

U-Boot

HANDS ON

1. Compilar U-Boot
2. Cargarlo en la SBC (Práctica II)
3. Interactuar con U-boot

Gracias.