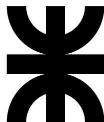
Programación en Linux embebido

Introducción a Linux embebido

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2017 –

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?

Introducción a Linux embebido

Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)

Introducción a Linux embebido

Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?

Introducción a Linux embebido

Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.
  - Portable: para diferentes arquitecturas de procesador (SoC): ARM, MISP, x86, PowerPC, etc.

Introducción a Linux embebido

Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.
  - Portable: para diferentes arquitecturas de procesador (SoC): ARM, MISP, x86, PowerPC, etc.
  - Open-source: libertad de descargar el código y modificarlo

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.
  - Portable: para diferentes arquitecturas de procesador (SoC): ARM, MISP, x86, PowerPC, etc.
  - Open-source: libertad de descargar el código y modificarlo
  - Comunidad activa: continuamente actualizado, soporte para nuevo Hw

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.
  - Portable: para diferentes arquitecturas de procesador (SoC): ARM, MISP, x86, PowerPC, etc.
  - Open-source: libertad de descargar el código y modificarlo
  - Comunidad activa: continuamente actualizado, soporte para nuevo Hw
- Desventaja: los sistemas complejos son más difíciles de entender

Introducción a Linux embebido
Sistema embebido

- ¿Qué es un sistema embebido?
  - Cualquier dispositivo que contiene una computadora pero que no luce como una computadora [¿?] (Ej.: lavarropas, televisores, impresora, robot, etc.)
  - Cuando estos dispositivos se hacen complejos surge la necesidad de contar con un SO $\longrightarrow$ Linux
- ¿Por qué utilizar Linux?
  - Funcionalidad: scheduler, stack de red, soporte USB, Wi-Fi, Bluetooth, almacenamiento, multimedia, etc.
  - Portable: para diferentes arquitecturas de procesador (SoC): ARM, MISP, x86, PowerPC, etc.
  - Open-source: libertad de descargar el código y modificarlo
  - Comunidad activa: continuamente actualizado, soporte para nuevo Hw
- Desventaja: los sistemas complejos son más difíciles de entender

(hay dos sistemas involucrados en la creación de un sistema embebido: host y target)

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain
- Bootloader: para cargar e iniciar (boot) el núcleo (kernel) de Linux

Introducción a Linux embebido

Elementos de un sistema embebido

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain
- Bootloader: para cargar e iniciar (boot) el núcleo (kernel) de Linux
- Kernel: corazón del SO, administra recursos y hace de interfaz con el Hw

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain
- Bootloader: para cargar e iniciar (boot) el núcleo (kernel) de Linux
- Kernel: corazón del SO, administra recursos y hace de interfaz con el Hw
- Root filesystem: contiene bibliotecas y programas

Elementos de un sistema embebido

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain
- Bootloader: para cargar e iniciar (boot) el núcleo (kernel) de Linux
- Kernel: corazón del SO, administra recursos y hace de interfaz con el Hw
- Root filesystem: contiene bibliotecas y programas

¿Qué más?

Cada proyecto comienza por obtener, personalizar, y poner en marcha los siguientes componentes:

- Toolchain: compilador y herramientas necesarias para crear el sistema para el dispositivo (target). Todo lo demás depende del toolchain
- Bootloader: para cargar e iniciar (boot) el núcleo (kernel) de Linux
- Kernel: corazón del SO, administra recursos y hace de interfaz con el Hw
- Root filesystem: contiene bibliotecas y programas

¿Qué más?

Colección de programas específicos para la aplicación embebida

Toolchain

Conjunto de herramientas para compilar el código fuente y obtener el binario o ejecutable que pueda correr en el dispositivo (target)

Toolchain

Conjunto de herramientas para compilar el código fuente y obtener el binario o ejecutable que pueda correr en el dispositivo (target)

Incluye:

- Compilador
- Linker (enlasador)
- Bibliotecas de tiempo de ejecución (run-time)

Introducción a Linux embebido
Toolchain

Conjunto de herramientas para compilar el código fuente y obtener el binario o ejecutable que pueda correr en el dispositivo (target)

Incluye:

- Compilador
- Linker (enlasador)
- Bibliotecas de tiempo de ejecución (run-time)

El toolchain GNU consiste de tres componentes:

- binutils: conjunto de binarios: ensamblador, linker y ld.

Introducción a Linux embebido
Toolchain

Conjunto de herramientas para compilar el código fuente y obtener el binario o ejecutable que pueda correr en el dispositivo (target)

Incluye:

- Compilador
- Linker (enlasador)
- Bibliotecas de tiempo de ejecución (run-time)

El toolchain GNU consiste de tres componentes:

- binutils: conjunto de binarios: ensamblador, linker y ld.
- GNU Compiler Collection (GCC): compilador de C y otros lenguajes (C++, Objective-C, Fortran, etc.)

Introducción a Linux embebido
Toolchain

Conjunto de herramientas para compilar el código fuente y obtener el binario o ejecutable que pueda correr en el dispositivo (target)

Incluye:

- Compilador
- Linker (enlasador)
- Bibliotecas de tiempo de ejecución (run-time)

El toolchain GNU consiste de tres componentes:

- binutils: conjunto de binarios: ensamblador, linker y ld.
- GNU Compiler Collection (GCC): compilador de C y otros lenguajes (C++, Objective-C, Fortran, etc.)
- Bibliotecas C: API estándar basado en la especificación POSIX

Introducción a Linux embebido

Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:

Introducción a Linux embebido
Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.

Introducción a Linux embebido

Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente

Introducción a Linux embebido

Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante

Introducción a Linux embebido
Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

GNU utiliza prefijos para identificar las combinaciones antes mencionadas

- CPU: arm, misp, x86_64, el (little-endian), eb (big-endian)

Introducción a Linux embebido
Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

GNU utiliza prefijos para identificar las combinaciones antes mencionadas

- CPU: arm, misp, x86_64, el (little-endian), eb (big-endian)
- Vendor: vendedor del toolchain, p.e.: buildroot, poky, unknown

Introducción a Linux embebido
Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

GNU utiliza prefijos para identificar las combinaciones antes mencionadas

- CPU: arm, misp, x86_64, el (little-endian), eb (big-endian)
- Vendor: vendedor del toolchain, p.e.: buildroot, poky, unknown
- Kernel: siempre será "linux"

Introducción a Linux embebido
Toolchain

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

GNU utiliza prefijos para identificar las combinaciones antes mencionadas

- CPU: arm, misp, x86_64, el (little-endian), eb (big-endian)
- Vendor: vendedor del toolchain, p.e.: buildroot, poky, unknown
- Kernel: siempre será "linux"
- SO: nombre del componente en espacio de usuario, será "gnu" o bien "uclibcgnu"

- El toolchain tiene que realizar la construcción de acuerdo del CPU target:
  - Arquitectura: ARM, MISP, x86_64, etc.
  - Operaciones en big o little -endian: algunas CPU pueden operar en ambos modos, pero el código es diferente
  - Soporte para punto flotante: si el CPU implementa unidad de punto flotante
  - Application Binary Interface (ABI): convensión para pasar parámetros a funciones (EABI, OABI, EABIHF)

GNU utiliza prefijos para identificar las combinaciones antes mencionadas

- CPU: arm, misp, x86_64, el (little-endian), eb (big-endian)
- Vendor: vendedor del toolchain, p.e.: buildroot, poky, unknown
- Kernel: siempre será “linux”
- SO: nombre del componente en espacio de usuario, será “gnu” o bien “uclibcgnu”

Ejemplos, **`$ gcc -dumpmachine`**: **`x86_64-linux-gnu`** (PC)

    mipsel-unknown-linux-gnu (libro)
    arm-cortex_a8-linux-gnueabihf (para QEMU)
    arm-linux-gnueabihf (Raspberry Pi)

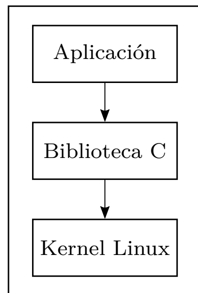
Toolchain – Biblioteca C e instalación

La biblioteca C es la implementación del estándar POSIX, y es la forma que tienen los programas de comunicarse con el kernel de Linux

Introducción a Linux embebido

Toolchain – Biblioteca C e instalación

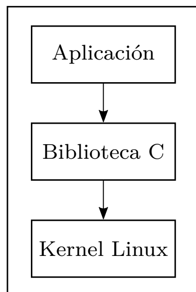
La biblioteca C es la implementación del estándar POSIX, y es la forma que tienen los programas de comunicarse con el kernel de Linux



Introducción a Linux embebido

Toolchain – Biblioteca C e instalación

La biblioteca C es la implementación del estándar POSIX, y es la forma que tienen los programas de comunicarse con el kernel de Linux

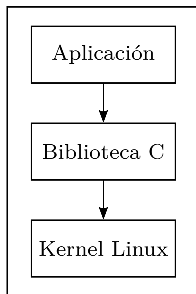


Biblioteca C disponibles

- glibc: biblioteca estándar C del proyecto GNU
- eglibs: biblioteca `glibc` embebida, inicialmente con parches, pero mergeado a partir de v2.20 (no mantenida)
- uClibc: implementación para microcontroladores (μC), inicialmente para μCLinux
- musl libc: nueva biblioteca C para sistemas embebidos

Toolchain – Biblioteca C e instalación

La biblioteca C es la implementación del estándar POSIX, y es la forma que tienen los programas de comunicarse con el kernel de Linux



Biblioteca C disponibles

- glibc: biblioteca estándar C del proyecto GNU
- eglibs: biblioteca `glibc` embebida, inicialmente con parches, pero mergeado a partir de v2.20 (no mantenida)
- uClibc: implementación para microcontroladores (μC), inicialmente para μCLinux
- musl libc: nueva biblioteca C para sistemas embebidos

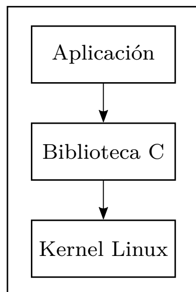
¿Cómo obtener el toolchain?

- Toolchain ya construido que coincida con nuestra necesidades
- Ya generado por alguna herramienta de construcción del sistema (Buildroot o Yocto Project)
- Crearlo uno mismo

Introducción a Linux embebido

Toolchain – Biblioteca C e instalación

La biblioteca C es la implementación del estándar POSIX, y es la forma que tienen los programas de comunicarse con el kernel de Linux



Biblioteca C disponibles

- glibc: biblioteca estándar C del proyecto GNU
- eglibs: biblioteca `glibc` embebida, inicialmente con parches, pero mergeado a partir de v2.20 (no mantenida)
- uClibc: implementación para microcontroladores (μC), inicialmente para μCLinux
- musl libc: nueva biblioteca C para sistemas embebidos

¿Cómo obtener el toolchain?

- Toolchain ya construido que coincida con nuestra necesidades
- Ya generado por alguna herramienta de construcción del sistema (Buildroot o Yocto Project)
- Crearlo uno mismo crosstool-NG

Introducción a Linux embebido

Toolchain – sysroot, bibliotecas y archivos headers

El `sysroot` del toolchain es el directorio cuyo subdirectorios contienen las bibliotecas, archivos headers, y otros archivos de configuración.

Introducción a Linux embebido

Toolchain – sysroot, bibliotecas y archivos headers

El `sysroot` del toolchain es el directorio cuyo subdirectorios contienen las bibliotecas, archivos headers, y otros archivos de configuración.

```
$ gcc -print-sysroot
```

Introducción a Linux embebido

Toolchain – sysroot, bibliotecas y archivos headers

El `sysroot` del toolchain es el directorio cuyo subdirectorios contienen las bibliotecas, archivos headers, y otros archivos de configuración.

```
$ gcc -print-sysroot
```

Dentro del `sysroot` se encuentra

- `lib`: objetos compartidos de biblioteca C y linker dinámico/cargador, `ld-linux`
- `usr/lib`: archivos de biblioteca estática C
- `usr/include`: archivos de cabecera de todas las bibliotecas
- `usr/bin`: programas de utilidad que se ejecuta sobre el target, tales como `ldd`
- `usr/share`: utilizado para localización e internacionalización
- `sbin`: utilidad `ldconfig` para cargar los path a las bibliotecas

Toolchain – sysroot, bibliotecas y archivos headers

El `sysroot` del toolchain es el directorio cuyo subdirectorios contienen las bibliotecas, archivos headers, y otros archivos de configuración.

```
$ gcc -print-sysroot
```

Dentro del `sysroot` se encuentra

- `lib`: objetos compartidos de biblioteca C y linker dinámico/cargador, `ld-linux`
- `usr/lib`: archivos de biblioteca estática C
- `usr/include`: archivos de cabecera de todas las bibliotecas
- `usr/bin`: programas de utilidad que se ejecuta sobre el target, tales como `ldd`
- `usr/share`: utilizado para localización e internacionalización
- `sbin`: utilidad `ldconfig` para cargar los path a las bibliotecas

Algunos son necesarios para compilar el programa en el desarrollo sobre el host, mientras que otros se necesitan en el target en tiempo de ejecución (?)

Introducción a Linux embebido

Toolchain – sysroot, bibliotecas y archivos headers

El `sysroot` del toolchain es el directorio cuyo subdirectorios contienen las bibliotecas, archivos headers, y otros archivos de configuración.

```
$ gcc -print-sysroot
```

Dentro del `sysroot` se encuentra

- `lib`: objetos compartidos de biblioteca C y linker dinámico/cargador, `ld-linux`
- `usr/lib`: archivos de biblioteca estática C
- `usr/include`: archivos de cabecera de todas las bibliotecas
- `usr/bin`: programas de utilidad que se ejecuta sobre el target, tales como `ldd`
- `usr/share`: utilizado para localización e internacionalización
- `sbin`: utilidad `ldconfig` para cargar los path a las bibliotecas

Algunos son necesarios para compilar el programa en el desarrollo sobre el host, mientras que otros se necesitan en el target en tiempo de ejecución (?)

Otras herramientas: `ar`, `as`, `cpp`, `gcov`, `gdb`, `gprof`, `ld`, `objcopy`, `objdump`, `readelf`, `size`

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. **`libc`:** parte principal que contiene las funciones más conocidas POSIX tales como **`printf`**, **`open`**, **`close`**, **`read`**, **`write`**, etc.

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. `libc`: parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. `libm`: funciones matemáticas tales como `cos`, `exp`, `log`, etc.

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. `libc`: parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. `libm`: funciones matemáticas tales como `cos`, `exp`, `log`, etc.
3. `libpthread`: funciones de threads POSIX cuyos nombres comienzan con `pthread_`

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. `libc`: parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. `libm`: funciones matemáticas tales como `cos`, `exp`, `log`, etc.
3. `libpthread`: funciones de threads POSIX cuyos nombres comienzan con `pthread_`
4. `librt`: extensión real-time de POSIX

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. **`libc`:** parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. **`libm`:** funciones matemáticas tales como `cos`, `exp`, `log`, etc.
3. **`libpthread`:** funciones de threads POSIX cuyos nombres comienzan con `pthread_`
4. **`librt`:** extensión real-time de POSIX

La primera (`libc`) se linkea siempre, pero las demás tiene que ser linkeadas de forma explícita con la opción `-l`.

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. `libc`: parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. `libm`: funciones matemáticas tales como `cos`, `exp`, `log`, etc.
3. `libpthread`: funciones de threads POSIX cuyos nombres comienzan con `pthread_`
4. `librt`: extensión real-time de POSIX

La primera (`libc`) se linkea siempre, pero las demás tiene que ser linkeadas de forma explícita con la opción `-l`.

Se puede verificar que biblioteca ha sido linkeada utilizando el comando `readelf`:

```
$ readelf -a app_bin | grep "Shared library"
```

Introducción a Linux embebido

Toolchain – componentes de la biblioteca C

La biblioteca C no es un único archivo. Está compuesta de cuatro partes principales que en conjunto implementan las funciones del API POSIX:

1. **`libc`**: parte principal que contiene las funciones más conocidas POSIX tales como `printf`, `open`, `close`, `read`, `write`, etc.
2. **`libm`**: funciones matemáticas tales como `cos`, `exp`, `log`, etc.
3. **`libpthread`**: funciones de threads POSIX cuyos nombres comienzan con `pthread_`
4. **`librt`**: extensión real-time de POSIX

La primera (`libc`) se linkea siempre, pero las demás tiene que ser linkeadas de forma explícita con la opción `-l`.

Se puede verificar que biblioteca ha sido linkeada utilizando el comando `readelf`:

```
$ readelf -a app_bin | grep "Shared library"
```

Las bibliotecas compartidas necesitan de un linker en tiempo de ejecución, el cual puedes ver haciendo:

```
$ readelf -a app_bin | grep ‘‘program interpreter’’
```

Introducción a Linux embebido

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

Introducción a Linux embebido

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

En un sistema embebido

1. inicialización básica del sistema
2. cargar el kernel de Linux

Introducción a Linux embebido

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

En un sistema embebido

1. inicialización básica del sistema
2. cargar el kernel de Linux

La secuencia de arranque consiste de tres etapas:

1. Código ROM: ROM en el SoC. Tiene acceso solo a la SRAM (eSRAM). Generalmente código propietario. Carga pequeñas porciones de código en la SRAM (desde FLASH NAND, eMMC, SD) o desde USB, UART, etc.

Introducción a Linux embebido

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

En un sistema embebido

1. inicialización básica del sistema
2. cargar el kernel de Linux

La secuencia de arranque consiste de tres etapas:

1. Código ROM: ROM en el SoC. Tiene acceso solo a la SRAM (eSRAM). Generalmente código propietario. Carga pequeñas porciones de código en la SRAM (desde FLASH NAND, eMMC, SD) o desde USB, UART, etc.
2. SPL (Secondary Program Loader): Configura el controlador de memoria para poder cargar el TPL en la memoria principal, DRAM. Generalmente, sin interacción con el usuario.

Introducción a Linux embebido

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

En un sistema embebido

1. inicialización básica del sistema
2. cargar el kernel de Linux

La secuencia de arranque consiste de tres etapas:

1. Código ROM: ROM en el SoC. Tiene acceso solo a la SRAM (eSRAM). Generalmente código propietario. Carga pequeñas porciones de código en la SRAM (desde FLASH NAND, eMMC, SD) o desde USB, UART, etc.
2. SPL (Secondary Program Loader): Configura el controlador de memoria para poder cargar el TPL en la memoria principal, DRAM. Generalmente, sin interacción con el usuario.
3. TPL (Third stage Program Loader): Cargador tipo U-Boot, Barebox, GRUB (GNU Grand Unified Bootloader)

Bootloader

Encargado de arrancar el sistema y cargar el kernel del SO.

En un sistema embebido

1. inicialización básica del sistema
2. cargar el kernel de Linux

La secuencia de arranque consiste de tres etapas:

1. Código ROM: ROM en el SoC. Tiene acceso solo a la SRAM (eSRAM). Generalmente código propietario. Carga pequeñas porciones de código en la SRAM (desde FLASH NAND, eMMC, SD) o desde USB, UART, etc.
2. SPL (Secondary Program Loader): Configura el controlador de memoria para poder cargar el TPL en la memoria principal, DRAM. Generalmente, sin interacción con el usuario.
3. TPL (Third stage Program Loader): Cargador tipo U-Boot, Barebox, GRUB (GNU Grand Unified Bootloader)

La Intel Galileo tiene arranque seguro y sigue el estándar UEFI (Uniersal Extensible Firmware Interface)

El kernel se encarga de tres tareas:

- administración de los recursos

Introducción a Linux embebido
Kernel

El kernel se encarga de tres tareas:

- administración de los recursos
- interacción con el hardware

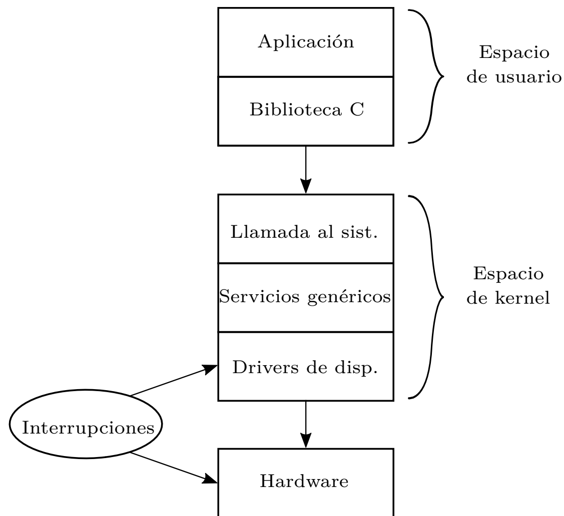
El kernel se encarga de tres tareas:

- administración de los recursos
- interacción con el hardware
- brindar a los programas en el espacio de usuario una API con un nivel de abstracción adecuado

Introducción a Linux embebido
Kernel

El kernel se encarga de tres tareas:

- administración de los recursos
- interacción con el hardware
- brindar a los programas en el espacio de usuario una API con un nivel de abstracción adecuado



Introducción a Linux embebido
Root filesystem

Al lanzar el kernel se le debe pasar un *sistema de archivo raíz*:

1. `ramdisk`
2. mediante un dispositivo de bloques ya montado

(parámetro `root=` de la línea de comando del kernel)

Introducción a Linux embebido

Root filesystem

Al lanzar el kernel se le debe pasar un *sistema de archivo raíz*:

1. `ramdisk`
2. mediante un dispositivo de bloques ya montado

(parámetro `root=` de la línea de comando del kernel)

Una vez que tiene un filesystem, el kernel puede ejecutar el primer programa, que tiene por defecto el nombre `init`.

Introducción a Linux embebido
Root filesystem

Al lanzar el kernel se le debe pasar un *sistema de archivo raíz*:

1. `ramdisk`
2. mediante un dispositivo de bloques ya montado

(parámetro `root=` de la línea de comando del kernel)

Una vez que tiene un filesystem, el kernel puede ejecutar el primer programa, que tiene por defecto el nombre `init`.

`init` ejecuta scripts de configuración, otros programas (llamanda a funciones del sistema de la biblioteca C), etc. Tiene PID=1 y se ejecuta con permisos del usuario `root`

Root filesystem

Al lanzar el kernel se le debe pasar un *sistema de archivo raíz*:

1. `ramdisk`
2. mediante un dispositivo de bloques ya montado

(parámetro `root=` de la línea de comando del kernel)

Una vez que tiene un filesystem, el kernel puede ejecutar el primer programa, que tiene por defecto el nombre `init`.

`init` ejecuta scripts de configuración, otros programas (llamanda a funciones del sistema de la biblioteca C), etc. Tiene PID=1 y se ejecuta con permisos del usuario `root`

Programas necesarios en el root filesystem

- `init`
- Shell
- Utilidades - BusyBox

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás

Componentes mínimos:

- **`init`**: programa encargado de ejecutar todo lo demás
- **`shell`**: le brinda al usuario un prompt de línea de comandos (útil a **`init`**)

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás
- `shell`: le brinda al usuario un prompt de línea de comandos (útil a `init`)
- `daemons`: programas de servicios, ejecutados por `init`

Introducción a Linux embebido
Root filesystem

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás
- `shell`: le brinda al usuario un prompt de línea de comandos (útil a `init`)
- `daemons`: programas de servicios, ejecutados por `init`
- `libraries`: generalmente los programas anteriores están linkeados de forma dinámica

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás
- `shell`: le brinda al usuario un prompt de línea de comandos (útil a `init`)
- `daemons`: programas de servicios, ejecutados por `init`
- `libraries`: generalmente los programas anteriores están linkeados de forma dinámica
- `Configuration files`: configuración para `init` y los demonios, archivos ASCII generalmente en `/etc`

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás
- `shell`: le brinda al usuario un prompt de línea de comandos (útil a `init`)
- `daemons`: programas de servicios, ejecutados por `init`
- `libraries`: generalmente los programas anteriores están linkeados de forma dinámica
- `Configuration files`: configuración para `init` y los demonios, archivos ASCII generalmente en `/etc`
- `Device nodes`: archivos especiales que brindan acceso de los drivers de dispositivos

Componentes mínimos:

- `init`: programa encargado de ejecutar todo lo demás
- `shell`: le brinda al usuario un prompt de línea de comandos (útil a `init`)
- `daemons`: programas de servicios, ejecutados por `init`
- `libraries`: generalmente los programas anteriores están linkeados de forma dinámica
- `Configuration files`: configuración para `init` y los demonios, archivos ASCII generalmente en `/etc`
- `Device nodes`: archivos especiales que brindan acceso de los drivers de dispositivos
- `/proc & /sysfs`: pseudo-sistemas de archivos que representan información estructurada del kernel

Componentes mínimos:

- **`init`**: programa encargado de ejecutar todo lo demás
- **`shell`**: le brinda al usuario un prompt de línea de comandos (útil a **`init`**)
- **`daemons`**: programas de servicios, ejecutados por **`init`**
- **`libraries`**: generalmente los programas anteriores están linkeados de forma dinámica
- **`Configuration files`**: configuración para **`init`** y los demonios, archivos ASCII generalmente en **`/etc`**
- **`Device nodes`**: archivos especiales que brindan acceso de los drivers de dispositivos
- **`/proc & /sysfs`**: pseudo-sistemas de archivos que representan información estructurada del kernel
- **`Kernel modules`**: generalmente en **`/lib/modules/[kernel version]`**

Introducción a Linux embebido

Root filesystem – esquema de directorios

- `/bin`: programas esenciales para todos los usuarios
- `/dev`: nodos de dispositivos u otros archivos especiales
- `/etc`: configuración del sistema
- `/lib`: bibliotecas compartidas esenciales, como por ejemplos la biblioteca C
- `/proc`: sistema de archivos (filesystem) `/proc`
- `/sbin`: programas esenciales para el administrador del sistema
- `/sys`: sistema de archivos (filesystem) `/sysfs`
- `/tmp`: para poner archivos temporales o volátiles
- `/usr`: cuanto mínimo, debería contener `/usr/bin`, `/usr/lib` y `/usr/sbin`, los cuales contienen programas adicionales, bibliotecas, y utilidades para la administración del sistema
- `/var`: jerarquía de archivos y directorios que puede ser modificado en tiempo de ejecución, por ejemplo, mensajes de log

Aprox. 50 utilidades para el sistema básico. Dos problemas:

1. Localizar el código fuente para cada uno y cross-compilarlo demanda un trabajo importante
2. La colección de programas resultantes ocupará varias decenas de megabytes, lo cual fue un problema en los primeros días de Linux embebido

Solución ⟶ `BusyBox`

Aprox. 50 utilidades para el sistema básico. Dos problemas:

1. Localizar el código fuente para cada uno y cross-compilarlo demanda un trabajo importante
2. La colección de programas resultantes ocupará varias decenas de megabytes, lo cual fue un problema en los primeros días de Linux embebido

Solución $\longrightarrow$ `BusyBox`

`BusyBox` combina todas las herramientas en un único binario

- Es una colección de applets. Nombre: `[applet]_main`
- Ej.: el comando `cat` está implementado en `coreutils/cat.c` y exporta `cat_main`
- La función principal de `BusyBox` ejecuta una llamada al applet correcto

Introducción a Linux embebido
Root filesystem – BusyBox

Aprox. 50 utilidades para el sistema básico. Dos problemas:

1. Localizar el código fuente para cada uno y cross-compilarlo demanda un trabajo importante
2. La colección de programas resultantes ocupará varias decenas de megabytes, lo cual fue un problema en los primeros días de Linux embebido

Solución ⟶ `BusyBox`

`BusyBox` combina todas las herramientas en un único binario

- Es una colección de applets. Nombre: `[applet]_main`
- Ej.: el comando `cat` está implementado en `coreutils/cat.c` y exporta `cat_main`
- La función principal de `BusyBox` ejecuta una llamada al applet correcto

Leer un archivo

```
$ busybox cat my_file.txt
```

Introducción a Linux embebido
Root filesystem – BusyBox

Aprox. 50 utilidades para el sistema básico. Dos problemas:

1. Localizar el código fuente para cada uno y cross-compilarlo demanda un trabajo importante
2. La colección de programas resultantes ocupará varias decenas de megabytes, lo cual fue un problema en los primeros días de Linux embebido

Solución ⟶ `BusyBox`

`BusyBox` combina todas las herramientas en un único binario

- Es una colección de applets. Nombre: `[applet]_main`
- Ej.: el comando `cat` está implementado en `coreutils/cat.c` y exporta `cat_main`
- La función principal de `BusyBox` ejecuta una llamada al applet correcto

Leer un archivo

```
$ busybox cat my_file.txt
```

(hacer `$ ls -l bin/cat bin/busybox`)