Marcobootstrap

Diego Martín Arroyo

Lunes, 11 de mayo de 2015

Resumen

Marcobootstrap comprende un conjunto de utilidades que permite llevar a cabo la descarga e instalación de un sistema operativo en varios nodos sin requerir la supervisión de un administrador, incluyendo el autodescubrimiento del servidor que aloja los ficheros binarios.

Índice

Introducción

El uso de un sistema conformado por varios nodos dificulta su mantenimiento, y en particular la instalación de un conjunto de herramientas iniciales y las posteriores actualizaciones. Dicho mantenimiento, si es llevado a cabo de forma "manual" suele propiciar fallos humanos. Por ello, en el desarrollo del sistema se ha apostado por una herramienta que automatice la instalación del sistema operativo reduciendo al máximo el tiempo de atención que el administrador debe prestar a cada nodo.
En el presente documento se detallan las decisiones de usuario llevadas a cabo en el desarrollo de la utilidad **marco-netinst**, que lleva a cabo este cometido, así como el funcionamiento de dicha herramienta.

Evaluación de alternativas

Como se define en la memoria del Trabajo, las opciones típicamente utilizadas por administradores de sistemas no son viables en el sistema a construir debido a las características de la plataforma utilizada: los equipos **Rasbperry Pi** no soportan el protocolo **PXE**, por lo que es necesario crear una herramienta propia, o buscar alternativas ya existentes creadas para un problema similar.

Estrategia inicial

El arranque de una placa **Rasbperry Pi** es llevado a cabo principalmente por la GPU de la placa. Al conectarse a la corriente eléctrica, se activa la secuencia de arranque definida en la memoria ROM de la placa. Este código busca en la primera partición de la tarjeta SD un *bootloader* y lo carga, activando la memoria SDRAM y pasando al siguiente estado, en el que se carga el archivo start.elf, encargado de cargar el Kernel del sistema operativo, según los parámetros definidos en los ficheros `config.txt`, `cmdline.txt` y **bcm2835.dtb**.

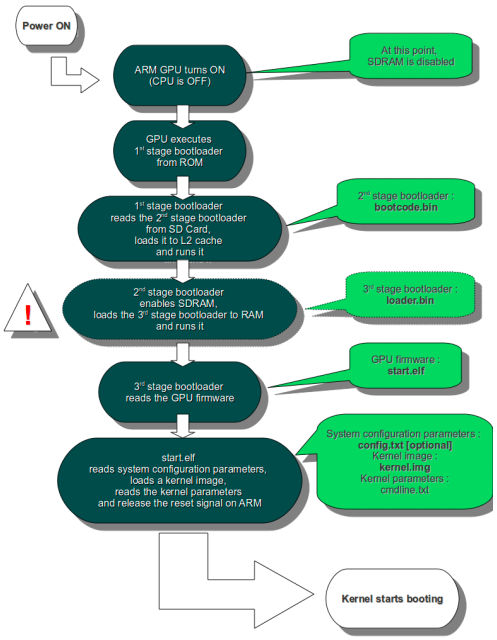

Figura 1: Secuencia de arranque de la Raspberry Pi[1]

Un archivo de configuración `cmdline.txt` sin modificar suele seguir la siguiente estructura:

```
root=/dev/mmcblk0p2 rw rootwait console=ttyAMA0,115200 console=tty1 \
selinux=0 plymouth.enable=0 smsc95xx.turbo_mode=N dwc_otg.lpm_enable=0 \
kgdboc=ttyAMA0,115200 elevator=noop
```

Los parámetros de interés son los siguientes:

- `root`: Partición raíz a utilizar (generalmente la segunda partición)
- `init`: Especifica el archivo a ejecutar en el arranque. Este proceso tendrá por tanto el **PID** 1 y generalmente su ejecutable es el fichero `/sbin/init`[1]. Si no se especifica, como ocurre en el archivo por defecto, su valor es este[2].

Modificando el parámetro `init` es posible ejecutar un código diferente a `init` en el proceso de arranque, que realice la descarga e instalación del sistema operativo, logrando una solución de compromiso entre la instalación manual y el uso de **PXE**.
Debido a la falta de experiencia en el desarrollo de herramientas similares, se han probado varios enfoques:

Primer enfoque: modificación del script de inicio

En este primer intento se crea un *script* y un ejecutable programado en C con un conjunto mínimo de utilidades que permitan la descarga del sistema operativo. Estos archivos son instalados en la primera partición de la tarjeta SD, siendo correctamente ejecutados. Sin embargo, no es posible llevar a cabo la instalación debido a la falta de herramientas de configuración de red.

Segundo enfoque: Busybox

Busybox es una utilidad que aglutina un conjunto de herramientas UNIX en un único ejecutable de tamaño muy reducido, reemplazando así el uso de varios paquetes, como fileutils o shellutils. Las utilidades a las que reemplaza cuentan con mucha menos funcionalidad, pero son capaces de crear un sistema utilizable con únicamente un kernel y varios ficheros de configuración[3]. Es la opción más común a la hora de crear sistemas embebidos o para realizar operaciones sin un sistema operativo.
Utilizando BusyBox es posible configurar la interfaz de red de la Raspberry Pi mediante **DHCP** gracias a la herramienta **udhcpc** así como a un conjunto de herramientas de utilidad como **wget**.

Incluyendo BusyBox en el código del primer proyecto es posible conseguir acceso a la red, pero el hecho de no utilizar el proceso **init** dificulta la mayor parte de operaciones, por lo que esta opción es descartada.

Enfoque definitivo

Antes de detallar la solución definitiva al problema, es necesario detallar las herramientas de terceros valoradas y descartadas:

Raspi-LTSP, PiNET

Estos proyectos[4, 5] están enfocados a la centralización de toda la información en un servidor, y delegan la carga del sistema operativo al mismo, por lo que no constituyen una opción viable.

[1] En sistemas donde `systemd` es el proceso inicial, `/sbin/init` es un enlace simbólico al ejecutable de `systemd`

BerryBoot

BerryBoot[6] es un sistema de arranque que posibilita la descarga e instalación de un sistema operativo y la coexistencia de varios en una misma tarjeta SD. Constituye una opción bastante prometedora, que sin embargo depende significativamente de una interfaz gráfica, haciendo que su uso sea inviable para el objetivo deseado.

Propuesta de solución

El proyecto **raspbian-ua-netinst**[7] posibilita la instalación de la distribución Raspbian desde un repositorio público, realizando el particionado de la tarjeta SD, la descarga e instalación de los ficheros y el resto de tareas necesarias desde la propia máquina, únicamente instalando en la tarjeta unos 15 megabytes de información. Este proyecto ha sido adaptado a las necesidades específicas del sistema, con las siguientes modificaciones:

- Instalación de **ArchLinux ARM** en lugar de **Raspbian**.
- Instalación del sistema operativo completo a partir de un archivo **.tar.gz** en lugar de la descarga de paquetes.
- Nuevo *script* de carga del *software* en la tarjeta SD (en el paquete original se delega a utilidades de terceros).
- Detección del servidor sin configuración previa utilizando **MarcoPolo**.

El sistema se compone de varios componentes, como se detalla en la siguiente sección.

marco-netinst

marco-netinst está compuesto por un conjunto de herramientas que permiten la gestión de los nodos del sistema y la instalación del sistema operativo en los mismos. Para ello se compone de varios módulos:

marco-netinst

Este módulo, basado en **raspbian-ua-netinst** únicamente comprende varios scripts, detallados a continuación:

update.sh

Descarga del repositorio indicado los paquetes que necesita el sistema operativo para arrancar. Dichos paquetes consisten básicamente en código de arranque, *kernels*, bibliotecas y un conjunto de herramientas básicas, siendo la más importante de ellas **BusyBox**. El *script* realiza la descarga en un directorio temporal de los archivos **.deb**.

build.sh

Crea los archivos necesarios para la instalación, que deberán ser copiados a la tarjeta SD. Para ello, realiza la siguiente secuencia de acción:

1. Copia del *kernel*.
2. Creación de los archivos de configuración.

3. Creación de los archivos **.cpio** (uno por cada arquitectura) que aglutina todas las bibliotecas, ejecutables y otros archivos de interés, así como los *scripts* que llevarán a cabo la instalación del sistema operativo.
4. Creación de los parámetros de la línea de comandos.
5. Copia de las utilidades de **MarcoPolo**.
6. Creación de un fichero **.zip**.

deploy.sh

Formatea la tarjeta SD, creando una única partición, donde descomprime el archivo **.zip** creado anteriormente.
Estos tres *scripts* deben ser ejecutados en el orden en el que han sido expuestos, pues dependen de los archivos generados por el resto. En caso de que no se realice la llamada de uno sin su antecesor, este procederá a su ejecución antes de realizar la funcionalidad que le corresponde.

clean.sh

Elimina todos los ficheros temporales y el resultado de las ejecuciones.

marco-bootstrap

Ejecutable en C++ que permite llevar a cabo el descubrimiento del servidor donde se aloja el sistema operativo mediante **MarcoPolo**. Debido a la ausencia del *daemon* de **Marco** en el sistema, se utiliza una implementación básica del protocolo, denominada **marco-minimal**. Esta implementación expone la misma funcionalidad que el **binding** de Marco en C++, pero a más bajo nivel.

marco-bootstrap-backend

Interfaz de gestión de los sistemas operativos que el servidor ofrece para su descarga.

Instalación

El conjunto de herramientas que realizan la instalación del sistema operativo están basadas en **Debian**, y por tanto utilizan el gestor de arranque **init**. Para realizar la instalación por tanto se delegará la ejecución de los *scripts* que realicen las funciones a dicho gestor. Por ello, se define el script `/etc/rcS`, que es ejecutado en cualquier nivel de ejecución (*runlevel*)[2].
El script lleva a cabo la siguiente secuencia de tareas:

1. Define una serie de variables de utilidad.
2. Crea de directorios necesarios dentro de la tarjeta SD.
3. "Instala" todas las utilidades de **BusyBox** (permite el acceso a las mismas) y define las rutas de búsqueda de ficheros mediante la variable `$PATH`.
4. Monta de los pseudosistemas de ficheros `/proc` y `/sys`.
5. Establece mecanismos de redirección de una copia de las salidas estándar y de error (**stdout** y **stderr**) a un fichero de log para su posterior análisis.

[2] apt-browse.org/browse/ubuntu/trusty/main/all/sysv-rc/2.88dsf-41ubuntu6/file/etc/rcS.d/README

6. Determina el tipo de *hardware* sobre el que se está ejecutando el *script*, a fin de instalar versión del sistema operativo apropiada.
7. Copia los ficheros de arranque contenidos en la tarjeta SD.
8. Carga un *script* con parámetros adicionales definidos por el usuario.
9. Configura de la interfaz de red **eth0** mediante **DHCP**.
10. Determina de la hora mediante **NTP**.
11. Carga módulos del *kernel* si son necesarios.
12. Particiona la tarjeta SD según el siguiente esquema:
    - Partición 1: 128 megabytes, formato FAT32.
    - Partición 2: Resto de la tarjeta SD, formato ext4.
13. Busca de servidores que alojen el sistema operativo mediante **MarcoPolo**.
14. Descarga el sistema operativo como archivo `.tar.gz`.
15. Descomprime el sistema operativo en la partición número 2 (donde la mayoría de ficheros se almacenan. La estructura del fichero `.tar.gz` define el lugar donde cada fichero se debe alojar, y los permisos de acceso a los ficheros son conservados en el proceso de extracción, por lo que con la extracción se consigue además la estructuración del sistema y la gestión de permisos).
16. Mueve los ficheros de arranque a la partición 1 (*kernel*, parámetros de arranque, etcétera).
17. Ejecuta los *scripts* de post-instalación.
18. Almacena el fichero de log.
19. Limpia archivos temporales.
20. Desmonta los sistemas de ficheros y procede al reinicio del sistema.

Una vez ejecutado el script el nodo está preparado para integrarse en el sistema. El tiempo total de instalación es de unos 5 minutos, según el fichero de log, dependiendo del estado de la red. En pruebas realizadas, el tiempo de descarga supone aproximadamente la mitad del tiempo total de instalación, si bien mejora significativamente cuando se cuenta con un servidor local que aloje el sistema operativo.

Referencias

[1] SG60, "What is the boot sequence?." `http://raspberrypi.stackexchange.com/questions/10442/what-is-the-boot-sequence`, nov 2013. Consultado: 2015-05-11.

[2] L. K. Organization, "Kernel parameters." `https://www.kernel.org/doc/Documentation/kernel-parameters.txt`.

[3] D. Vlasenko, "About BusyBox." `http://www.busybox.net/about.html`, mar 2015. Consultado: 2015-05-11.

[4] A. Mulholland, "RaspberryPi-LTSP." `https://github.com/gbaman/RaspberryPi-LTSP`, apr 2015. Consultado: 2015-05-11.

[5] A. Mulholland, "PiNet." `http://pinet.org.uk/`, apr 2015. Consultado: 2015-05-11.

[6] F. Bos, "BerryBoot v2.0 - bootloader / universal operating system installer." `http://www.berryboot.com/`, mar 2015. Consultado: 2015-05-11.

[7] Rasbpian, "raspbian-ua-netinst." `https://github.com/debian-pi/raspbian-ua-netinst`, May 2015.

[8] eLinux.org, "Rpiconfig - elinux.org." http://elinux.org/RPiconfig. Consultado: 2015-05-11.