Trabajo Práctico
Implementación de Booteo con Ext2 en Sodium

Ariel Cacho Mendoza, Pablo Penalba, Diego Stallo

Dpto. de Ingeniería e Investigaciones Tecnológicas, Universidad Nacional de La Matanza,
Florencio Varela 1903 C.P. 1754 San Justo, Argentina
pavlo.penalba@gmail.com, dstallo@gmail.com, arielcm@gmail.com

Resumen. En este trabajo se muestra la investigación y análisis sobre el procedimiento de booteo del sistema operativo SODIUM y el sistema de archivos EXT2. El objetivo es definir el marco teórico necesario para la implementación de booteo con EXT2 en SODIUM.
Palabras Clave: sistema de archivos extendido, EXT2, superbloque, i-nodo, bootstrap, BIOS, Partition Boot Sector, Loader

1. Introducción

El propósito de este documento es detallar la investigación efectuada para la implementación del booteo del sistema operativo académico SODIUM, a partir de un sistema de archivos EXT2. En primera instancia, se indagará acerca del procedimiento de booteo general de un LINUX, desde el arranque de la computadora, y luego se entrará en detalle en el procedimiento de booteo del SODIUM, obtenido mediante su código fuente y documentación. Actualmente, el SODIUM puede configurarse para bootear desde los sistemas de archivos FAT12, FAT16 y FAT32, y se pretende incluir EXT2, sin modificar los procedimientos en los otros sistemas de archivos.

En segundo lugar, se procederá a investigar el sistema de archivos EXT2, características centrales e implementación, con el objetivo de comprender la estructura que deberá ser leída para obtener los archivos necesarios para levantar a memoria.

2. Procedimiento de Booteo - Bootstrap

A continuación se dará una descripción del procedimiento de booteo básico de un sistema operativo LINUX y también se indagará en las particularidades del sistema operativo SODIUM. La investigación se centrará en qué es lo que ocurre desde que se arranca la computadora, hasta que el *Kernel*[1] del SO es cargado a memoria y toma el

[1] El kernel es la parte más importante de un sistema operativo. Es el principal responsable de permitir a los distintos programas el acceso al hardware de la computadora, a través de las denominadas "System Calls" o llamadas al sistema, y también es el encargado de gestión de los recursos de la computadora.

control de la ejecución. Este proceso recibe el nombre de *bootstrap*.

2.1. BIOS

El procedimiento de *bootstrap* o *bootstrapping* comienza desde el momento en que se presiona el botón de encendido de la PC[2] y un circuito de hardware específico, activa el pin de RESET de la CPU (enviando el valor lógico "1"). En este punto, la memoria principal de la computadora (RAM) y los registros del procesador, se encuentran con valores aleatorios sin ninguna utilidad. Luego del RESET, algunos registros del procesador toman valores prefijados y comienza la ejecución del código que se encuentra en la dirección física de memoria 0xfffffff0. Esta dirección es mapeada (es decir, traducida) hacia un dispositivo de memoria persistente de sólo lectura, que recibe el nombre de ROM. El código que se ejecutará corresponde a un conjunto de programas que se denomina BIOS[3] e incluye un conjunto de rutinas de bajo nivel, que suelen ser utilizadas por algunos sistemas operativos para controlar el hardware de la computadora (como por ejemplo MS-DOS).

Los sistemas operativos Linux utilizan las rutinas de la BIOS sólo en la fase de bootstrapping para obtener el kernel desde el disco o desde algún otro dispositivo externo. Una vez inicializado, Linux no utiliza más la BIOS, sino que provee sus propios *drivers*[4] para cada dispositivo de hardware en la computadora.

La BIOS esencialmente realiza cuatro tareas:

1. En primer lugar, se realiza una serie de pruebas sobre el hardware de la computadora, cuyo objetivo es la verificación de los distintos dispositivos, para comprobar si funcionan correctamente. Esta fase se denomina POST, que es un acrónimo en inglés de Power On Self Test (auto-prueba de encendido).
2. La segunda tarea es la inicialización los dispositivos de hardware, y esta fase es indispensable para garantizar que los dispositivos operen sin conflictos entre sí, sobre las líneas de IRQ[5] y los puertos de entrada y salida.
3. En tercera instancia, la BIOS busca un sistema operativo para iniciar. Para hacerlo, accederá en un orden que puede ser configurable, al primer sector de

---

[2] El bootstrapping de un sistema depende en gran medida de la arquitectura de la computadora. La presente investigación se basará en la arquitectura IBM, que es la más frecuentemente utilizada.

[3] BIOS: Basic Input Output System (Sistema básico de entrada y salida).

[4] Un controlador o *driver* es un programa informático que permite al sistema operativo interactuar con un dispositivo de hardware, proporcionando una interfaz para poder utilizarlo.

[5] Las líneas de IRQ son las rutas que se les asignan a cada dispositivo para enviar las Interrupciones al procesador. Las interrupciones son señales que indican al procesador que debe "interrumpir" el curso de ejecución actual, para ejecutar un código específico de atención a la interrupción. Este es un mecanismo que tienen los dispositivos periféricos para enviarle información al procesador, sin necesidad de esperar a ser "sondeado" por el mismo.

determinados dispositivos como pueden ser disqueteras, cd-rom, discos duros o periféricos usb (como puede ser un pen drive), para reconocer si dicho dispositivo es booteable o no.

4. Una vez que halló un dispositivo válido de booteo, la BIOS copiará a la memoria principal, en la dirección 0x00007c00, el primer sector de dicho dispositivo, y pondrá en ejecución dicho contenido, que deberá ser el código necesario para iniciar la carga del sistema operativo.

Para determinar si un disquete es booteable o no, la BIOS debe hallar una marca en los últimos 2 bytes de su primer sector. Dicha marca es el código 0xaa55, y cuando la BIOS la encuentra es cuando procederá a copiar el resto del contenido del primer sector a memoria e iniciar su ejecución.

2.2. Booteo de un LINUX desde un disco duro.

Sin embargo, el booteo desde un disco duro es algo diferente. El primer sector del disco duro es conocido como MBR (Master boot record – Registro de arranque principal) e incluye una tabla de particiones[6] y un pequeño programa, que se encargará de copiar el código del primer sector de la partición que contiene el sistema operativo a cargar (conocido como Partition Boot Sector, el cuál también deberá poseer la marca 0xaa55 en los últimos 2 bytes del sector).

Para identificar qué partición se deberá elegir para bootear existen distintas estrategias. Una de ellas (la utilizada por el sistema operativo SODIUM y también por Microsoft Windows) identifica la partición utilizando sencillamente una bandera con un valor lógico activo. Linux, en cambio, reemplaza el código sencillo ubicado en el MBR, por un programa más complejo que permite al usuario seleccionar la partición a bootear. Este programa recibió el nombre de LILO (LInux LOader), aunque actualmente, el programa más comúnmente utilizado pasó a ser el GRUB. LILO (y también GRUB) además ofrece seleccionar qué imagen del kernel de Linux se desea iniciar y una vez seleccionada, procederá a copiar dicha imagen a memoria y también un código específico de inicialización de Linux que se denomina Setup. Dicha rutina se encargará de inicializar los dispositivos de hardware y preparar el sistema para la ejecución del kernel (aunque la BIOS ya inició la mayoría de los dispositivos, Linux los reinicializa a su manera). La rutina de setup descomprime la imagen del kernel y finaliza cediendo la ejecución a la función start_kernel(), la cuál completará el procedimiento de booteo.

2.3. Booteo del SODIUM: El módulo SOLO.

---

[6] Una partición de disco es el nombre que se le da a cada división lógica que se encuentra en un solo dispositivo físico de almacenamiento (como puede ser un disco duro). Cada partición (un disco puede tener una sola partición) debe poseer su propio sistema de archivos (básicamente, el formato en el cuál se guardarán los archivos a almacenar). La tabla de particiones contendrá información básica sobre cada partición creada en el dispositivo (cada entrada usualmente posee el sector inicial y final de la partición y el tipo de sistema operativo que la controla).

Dejando de lado el procedimiento general de los Linux, ahora se indagará en el específico del SODIUM, desde que el programa contenido en el MBR copia el primer sector de la partición en la que se encuentra el SO (con la marca 0xaa55) y lo pone en ejecución. En este punto, el módulo que entra en juego es el llamado SOLO (SOdium LOader). El mismo consta de 2 rutinas básicas: el PBS y el Loader, de las cuales se describirán sus funciones principales.

PBS: Este es el código que se encuentra copiado en el primer sector de la partición, y es lo primero en ejecutarse del SODIUM. Su función es la de buscar el resto del módulo SOLO (el programa LOADER), dentro de la partición del SODIUM. Luego de encontrarlo, lo cargará a memoria y cederá la ejecución al mismo. Esto es de esta manera, dado que únicamente el programa del MBR (o en el caso de booteo desde floppy disk, la BIOS) copia a memoria un solo sector del disco, con un tamaño de 512 bytes, y por lo tanto, el código que se puede incluir es demasiado reducido.

Loader: Su función es la de cargar todos los archivos del SODIUM a memoria (actualmente se cargan a memoria todos los archivos y programas del SODIUM debido a que todavía no se encuentran finalizados los drivers propios del SODIUM, para controlar los dispositivos de almacenamiento y file system. Una vez que se hayan finalizado, se deberá cargar únicamente el kernel de SODIUM). Para realizarlo, busca en el dispositivo el archivo LISTADO.BIN, el cuál contiene la lista de los archivos a cargar, junto con las posiciones de memoria en las cuales deben ser cargados (este archivo es generado en momento de compilación del SODIUM).

El Loader carga a memoria el LISTADO.BIN y luego procederá a cargar todos los archivos en las direcciones indicadas. Por último, cederá la ejecución al Sodium.sys

3. Sistema de Archivos EXT2

El sistema de archivos extendido (Extended File System) fue introducido por Rémy Card en abril de 1992, como el primer sistema de archivos específicamente creado para el Kernel de Linux, en reemplazo del sistema de archivos de MINIX. Fue la primer implementación que utilizó VFS (Virtual File System)[7] y podía manejar sistemas de archivos de hasta 2 GB.

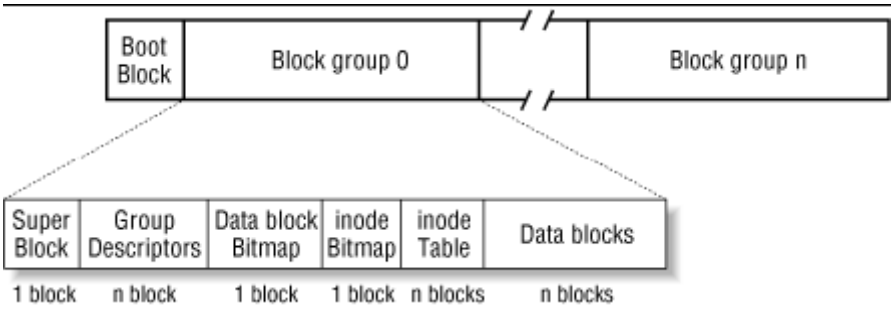
En enero de 1993, el mismo Rémy Card introdujo el sistema de archivos EXT2 (Second Extended File System), como solución a determinados problemas que tenía EXT: no tenía soporte para múltiples timestamps separados de acceso a archivos, no permitía modificar los inodos (estructura de información que se verá a continuación) y modificación de datos. EXT2 fue diseñado teniendo cuenta posibles modificaciones futuras, y por eso en muchas de sus estructuras hay espacio reservado para posibles nuevos campos de información (que se fueron incorporando en las versiones

[7] VFS es una interfaz entre el kernel de un sistema operativo y el sistema de archivos, que genera una capa de abstracción que permite a las aplicaciones acceder de manera uniforme, a distintos tipos de sistemas de archivos concretos. Por ejemplo, VFS puede ser usado para acceder a dispositivos de almacenamiento locales y remotos, de manera transparente sin que una aplicación note la diferencia.

consecuentes, EXT3 y EXT4).

3.1. Estructura General

El espacio de una partición formateada en EXT2, se encuentra divido en unidades básicas lógicas que se denominan bloques. El tamaño de estos bloques es fijo y no necesariamente coincide con el tamaño de los sectores del dispositivo en el que se almacena. Puede ser 1024, 2048 o 4096 bytes, valor que se selecciona al momento del formateo y no puede cambiarse posteriormente. Los bloques, a su vez, se agrupan en los llamados Grupos de Bloques (Block Groups), de una cantidad fija de bloques cada uno, también determinada al momento del formateo, de acuerdo al tamaño de la partición y al tamaño de un bloque. Cada bloque, además, tiene asignado un número que se utiliza como puntero para referenciar al bloque en cuestión.
La estructura de información más importante de EXT2 se denomina I-Nodo. La misma es una estructura de usualmente 128 bytes que contiene información sobre un archivo o un directorio dentro del sistema de archivos. Cada grupo de bloques agrupa a su vez una cantidad fija de inodos reservados, que referenciarán a los distintos archivos de la partición. Más adelante, se profundizará en la investigación de los I-Nodos, los cuales son indispensables para comprender el funcionamiento de EXT2.
El siguiente gráfico detalla la estructura básica de un sistema de archivos EXT2:



[8]

El primer elemento que figura en el gráfico es el llamado Boot Block (Bloque de booteo). Este es sencillamente un bloque reservado para el sector de booteo de la partición (PBS: mencionado en el punto 2.2), y no es manejado por el administrador de archivos. A continuación y de manera contigua, aparecerán los Grupos de Bloques. Cada uno de ellos presenta la misma división lógica, detallada en los puntos siguientes:

3.2. Superbloque

El superbloque (Super Block) es un bloque utilizado para almacenar información

[8] Bovet, D.P, Cesati M.: Understanding the Linux Kernel. O'Reilly. p 453. (2000)

acerca de toda la partición EXT2. Tanto el superbloque como los Descriptores de Grupo (Group Descriptors), contienen la misma información repetida en cada uno de los grupos de bloques. Esto es así, dado que contienen información indispensable para el funcionamiento del sistema de archivos y si se llegase a perder, todo el sistema de archivos queda inutilizable. Por ello, cada grupo de bloques contiene copias de seguridad del superbloque de los descriptores de grupos.

Los campos de información contenidos en el superbloque y su tamaño, se pueden obtener de la siguiente tabla informativa:

_ _u32	s_inodes_count	Total number of inodes
_ _u32	s_blocks_count	Filesystem size in blocks
_ _u32	s_r_blocks_count	Number of reserved blocks
_ _u32	s_free_blocks_count	Free blocks counter
_ _u32	s_free_inodes_count	Free inodes counter
_ _u32	s_first_data_block	Number of first useful block (always 1)
_ _u32	s_log_block_size	Block size
_ _s32	s_log_frag_size	Fragment size
_ _u32	s_blocks_per_group	Number of blocks per group
_ _u32	s_frags_per_group	Number of fragments per group
_ _u32	s_inodes_per_group	Number of inodes per group
_ _u32	s_mtime	Time of last mount operation
_ _u32	s_wtime	Time of last write operation
_ _u16	s_mnt_count	Mount operations counter
_ _u16	s_max_mnt_count	Number of mount operations before check
_ _u16	s_magic	Magic signature
_ _u16	s_state	Status flag
_ _u16	s_errors	Behavior when detecting errors
_ _u16	s_minor_rev_level	Minor revision level
_ _u32	s_lastcheck	Time of last check
_ _u32	s_checkinterval	Time between checks
_ _u32	s_creator_os	OS where filesystem was created
_ _u32	s_rev_level	Revision level
_ _u16	s_def_resuid	Default UID for reserved blocks
_ _u16	s_def_resgid	Default GID for reserved blocks
_ _u32	s_first_ino	Number of first nonreserved inode
_ _u16	s_inode_size	Size of on-disk inode structure
_ _u16	s_block_group_nr	Block group number of this superblock
_ _u32	s_feature_compat	Compatible features bitmap
_ _u32	s_feature_incompat	Incompatible features bitmap

[9]

3.3. Descriptores de Grupos

Un descriptor de grupo (Group Descriptor) es una estructura de 24 bytes, que contiene información sobre un grupo de bloques. Hay un descriptor por cada grupo de bloques y cada descriptor hace referencia al grupo de acuerdo al índice de ordenamiento que posee. Al igual que el superbloque, el array de descriptores de grupo posee copias de

[9] Ídem. p 455.

seguridad en cada uno de los grupos de bloques.

Type	Field	Description
__u32	bg_block_bitmap	Block number of block bitmap
__u32	bg_inode_bitmap	Block number of inode bitmap
__u32	bg_inode_table	Block number of first inode table block
__u16	bg_free_blocks_count	Number of free blocks in the group
__u16	bg_free_inodes_count	Number of free inodes in the group
__u16	bg_used_dirs_count	Number of directories in the group
__u16	bg_pad	Alignment to word
__u32 [3]	bg_reserved	Nulls to pad out 24 bytes

[10]

3.4. Mapa de bits de bloques de datos (Data block bitmap)

Este es un bloque destinado a ser un bitmap que indica el estado (libre u ocupado) de los bloques de datos del grupo de bloques (los bloques de datos son los bloques destinados a almacenar los archivos y directorios de la partición). Este bitmap necesariamente debe ocupar 1 sólo bloque como máximo, con lo cuál la cantidad de bloques de datos en un grupo de bloques, no puede superar el tamaño en bits de un bloque (por ejemplo, si los bloques son de 1024 bytes, entonces la cantidad de bloques de datos en un grupo de bloques, será de 8192).

3.5. Mapa de bits de inodos (Inode bitmap).

Esta estructura también es necesariamente de cómo máximo 1 bloque e indica el estado (libre u ocupado) de cada uno de los inodos dentro del grupo de bloques.

3.6. Tabla de Inodos

La tabla de inodos consiste en una sucesión de todos los inodos del grupo de bloques, estén ocupados o libres (se reserva el espacio para la cantidad máxima de inodos de un grupo de bloques). Es una serie de bloques consecutivos, cada uno de los cuales contiene un número predefinido de inodos (según el tamaño de un bloque).

Cada inodo (al igual que cada bloque) posee un número asignado, que permite ubicarlo de acuerdo a su posición dentro de un grupo de bloques. Entre la información disponible en un inodo, figura el tipo de archivo, owner, permisos del archivo, tamaño y timestamps de modificación del archivo.

A continuación se incluye la estructura de campos de un inodo:

[10] Ídem. p 456.

Type	Field	Description
__u16	i_mode	File type and access rights
__u16	i_uid	Owner identifier
__u32	i_size	File length in bytes
__u32	i_atime	Time of last file access
__u32	i_ctime	Time that inode last changed
__u32	i_mtime	Time that file contents last changed
__u32	i_dtime	Time of file deletion
__u16	i_gid	Group identifier
__u16	i_links_count	Hard links counter
__u32	i_blocks	Number of data blocks of the file
__u32	i_flags	File flags
union	osd1	Specific operating system information
__u32 [EXT2_N_BLOCKS]	i_block	Pointers to data blocks
__u32	i_version	File version (for NFS)
__u32	i_file_acl	File access control list
__u32	i_dir_acl	Directory access control list
__u32	i_faddr	Fragment address
union	osd2	Specific operating system information

11

Dentro del inodo, se encuentra un array de 15 punteros a bloques de datos, que recibe el nombre de *i_block,* y que apuntan a la información propia del archivo. Los primeros 12 punteros, referencian a bloques de datos directos del archivo; en cambio, los últimos 3 punteros apuntan a bloques de indirección. El primero de estos, se denomina puntero de indirección simple. Este puntero, apunta a un bloque de datos el cuál posee a su vez un array de punteros a bloques, los cuales poseerán el contenido propio del archivo (la cantidad de punteros en este bloque, dependerá de su tamaño. Cada puntero es un número entero de 4 bytes, con lo cuál si el tamaño del bloque es de 1024 bytes, el bloque de indirección poseerá 256 punteros). El segundo puntero indirecto, se denomina de indirección doble, dado que apunta a un bloque de punteros de indirección simple; es decir, que los punteros del bloque a su vez apuntan a otros bloques de punteros y recién estos finalmente referencian a los bloques de datos del archivo. El tercer puntero indirecto, es de indirección triple, es decir, que apunta a un bloque de punteros de indirección doble. Esta estructura de información, permite almacenar archivos de hasta 2TB de tamaño.

3.7. Bloques de datos (Data Blocks)

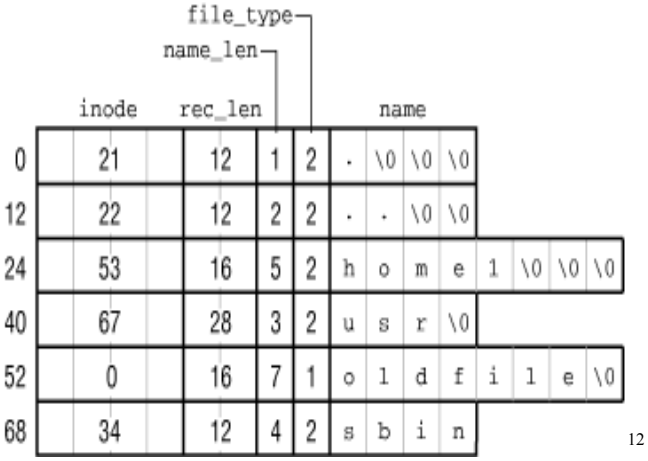
Por último, dentro del grupo de bloques, se encuentran los bloques en los cuáles se almacenarán los archivos y directorios propios del sistema de archivos. A estos bloques de datos, es a los cuáles apuntan los inodos de la tabla de inodos, dado que un inodo siempre apunta a bloques de datos del mismo grupo de bloques.

---

[11] Ídem. p 457.

3.8. Directorios

En EXT2 todas las estructuras de datos se representan con inodos. No solamente los archivos regulares, sino también los named pipes, sockets, dispositivos e incluso los links simbólicos. Los directorios en EXT2 también son un tipo especial de archivo, cuyos bloques de datos contendrán todas las entradas de directorio (es decir los archivos que contiene el directorio) que el mismo posee. La estructura de información en los bloques de datos de los directorios es la siguiente:



[12]

Como se ve, cada entrada de directorio posee el número de inodo (que se utilizará para encontrar el inodo dentro del sistema de archivos), el tipo de archivo y el nombre del archivo. Este último siempre ocupará un tamaño múltiplo de 4 bytes, rellenándose con NULLs al final si correspondiere. Todos los directorios tienen por defecto las entradas . y .., que apuntan al inodo de ese mismo directorio y al inodo del directorio padre (el directorio raíz no posee directorio padre, con lo cuál la entrada .. también apunta a sí mismo)

4. Conclusión:

Mediante la investigación realizada pudimos ver los pasos de inicialización de un sistema operativo y las estructuras necesarias para reconocer y leer un sistema de archivos EXT2. Esta investigación, resulta fundamental para determinar las modificaciones necesarias para poder bootear SODIUM desde un sistema de archivos EXT2.

[12] Ídem. p 459.

5. Bibliografía

1. Sitio Oficial EXT2, http://lxr.free-electrons.com/source/include/linux/ext2_fs.h#L374
2. Design and Implementation of the Second Extended Filesystem http://www.win.tue.nl/~aeb/linux/fs/ext2/ext2intro.html
3. The Second Extended File System, http://www.win.tue.nl/~aeb/linux/fs/ext2/ext2.html
4. Bovet, D.P, Cesati M.: Understanding the Linux Kernel. O'Reilly. (2000)
5. OSDev Wiki: EXT2, http://wiki.osdev.org/Ext2
6. Wikipedia: Extended File System http://en.wikipedia.org/wiki/Extended_file_system
7. Wikipedia: Ext2, http://en.wikipedia.org/wiki/Ext2
8. Wikipedia: Virtual File System, http://en.wikipedia.org/wiki/Virtual_file_system
9. Wikipedia: Partición de Disco, http://es.wikipedia.org/wiki/Particion_de_disco
10. Wikipedia: Interrupción, http://es.wikipedia.org/wiki/Interrupcion
11. Wikipedia: Power-on Self-test, http://en.wikipedia.org/wiki/Power-on_self-test
12. Wikipedia: Kernel (Computing), http://en.wikipedia.org/wiki/Kernel_%28computing%29
13. Wikipedia: Controlador de dispositivos, http://es.wikipedia.org/wiki/Controlador_de_dispositivo