Chapitre 14

Le cas du démarrage de Linux

Nous avons vu comment passer au mode 64 bits à partir du démarrage d'un microprocesseur x86-64. Voyons comment cela est effectué pour le système d'exploitation Linux, pour un noyau 64 bits.

14.1 Les moyens d'étude

Sources.- Les sources du noyau Linux sont livrées avec toute distribution GNU/Linux. On peut aussi les retrouver sur le site officiel de Linux :

`https://github.com/torvalds/linux`

ainsi que sur le site :

`https://www.kernel.org/`

On peut étudier ce qui se passe pour Linux à partir de ceux-ci.

Code.- On peut également préférer regarder ce qui se passe sur « notre » système Linux. Pour cela, il suffit de rechercher, dans le répertoire `/boot/` un fichier dont le nom commence par `vmlinuz`, d'une taille de quelques MiO. Il peut y en avoir plusieurs si on permet de démarrer avec plusieurs versions de Linux pas très éloignées les unes des autres : en général il s'agira de celle de la version la plus élevée ; de toutes façons, la version n'a pas beaucoup d'importance pour ce que nous allons étudier.

Pour notre part, nous étudierons `vmlinuz64-3.4.42` de 5 MiO.

Visualiseur hexadécimal.- Ne s'agissant pas d'un fichier texte, un éditeur de texte ne donne rien d'intéressant. On peut, par contre, utiliser un visualiseur hexadécimal, tel que `hexdump` :

```
$ hexdump -n 200 vmlinuz64-3.4.42
0000000 05ea c000 8c07 8ec8 8ed8 8ec0 31d0 fbe4
0000010 befc 002d 20ac 74c0 b409 bb0e 0007 10cd
0000020 f2eb c031 16cd 19cd f0ea 00ff 44f0 7269
0000030 6365 2074 6f62 746f 6e69 2067 7266 6d6f
0000040 6620 6f6c 7070 2079 7369 6e20 206f 6f6c
0000050 676e 7265 7320 7075 6f70 7472 6465 0d2e
0000060 500a 656c 7361 2065 7375 2065 2061 6f62
0000070 746f 6c20 616f 6564 2072 7270 676f 6172
0000080 206d 6e69 7473 6165 2e64 0a0d 520a 6d65
0000090 766f 2065 6964 6b73 6120 646e 7020 6572
00000a0 7373 6120 796e 6b20 7965 7420 206f 6572
00000b0 6f62 746f 2e20 2e20 2e20 0a0d 0000 0000
00000c0 0000 0000 0000 0000
00000c8
$
```

Éditeur hexadécimal.- On peut aussi utiliser un éditeur hexadécimal tel que `hexedit`, ici pour `vmlinuz-3.0.0-12-generic` :

```
00000000 EA 05 00 C0  07 8C C8 8E  D8 8E C0 8E  D0 31 E4 FB  FC BE 2D 00  AC 20 C0 74  .............1....-.. .t
00000018 09 B4 0E BB  07 00 CD 10  EB F2 31 C0  CD 16 CD 19  EA F0 FF 00  F0 44 69 72  ..........1..........Dir
00000030 65 63 74 20  62 6F 6F 74  69 6E 67 20  66 72 6F 6D  20 66 6C 6F  70 70 79 20  ect booting from floppy
00000048 69 73 20 6E  6F 20 6C 6F  6E 67 65 72  20 73 75 70  70 6F 72 74  65 64 2E 0D  is no longer supported..
00000060 0A 50 6C 65  61 73 65 20  75 73 65 20  61 20 62 6F  6F 74 20 6C  6F 61 64 65  .Please use a boot loade
00000078 72 20 70 72  6F 67 72 61  6D 20 69 6E  73 74 65 61  64 2E 0D 0A  0A 52 65 6D  r program instead....Rem
00000090 6F 76 65 20  64 69 73 6B  20 61 6E 64  20 70 72 65  73 73 20 61  6E 79 20 6B  ove disk and press any k
000000A8 65 79 20 74  6F 20 72 65  62 6F 6F 74  20 2E 20 2E  20 2E 0D 0A  00 00 00 00  ey to reboot . . .......
000000C0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
000000D8 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
000000F0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000108 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000120 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000138 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000150 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000168 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000180 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00000198 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
000001B0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
000001C8 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
000001E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 21 01 00  FB 6C 04 00  .................!...l..
000001F8 00 00 FF FF  01 08 55 AA  EB 62 48 64  72 53 0A 02  00 00 00 00  00 10 C4 36  ......U..bHdrS.........6
```

Il est évidemment difficile d'interpréter ce code à la volée. Remarquons que le premier secteur se termine bien par la signature `0x55 0xAA` aux décalages `510` et `511`, soit `0x1FE` et `0x1FF`.

Désassemblage.- Notre dernier fichier a une taille de plus de 4 MiO (exactement 4 658 096 octets). N'en conservons que les deux premiers « secteurs » pour obtenir un fichier plus léger :

```
$ dd if=./vmlinuz-3.0.0-12-generic of=./sect1.com bs=1024 count=1
```

On peut le désassembler en utilisant des options de `objdump` :

```
$ objdump -D -b binary -mi8086 ./sect1.com

./sect1.com:     file format binary


Disassembly of section .data:

00000000 <.data>:
   0:   ea 05 00 c0 07          ljmp   $0x7c0,$0x5
   5:   8c c8                   mov    %cs,%ax
   7:   8e d8                   mov    %ax,%ds
   9:   8e c0                   mov    %ax,%es
   b:   8e d0                   mov    %ax,%ss
   d:   31 e4                   xor    %sp,%sp
   f:   fb                      sti
  10:   fc                      cld
  11:   be 2d 00                mov    $0x2d,%si
  14:   ac                      lods   %ds:(%si),%al
  15:   20 c0                   and    %al,%al
  17:   74 09                   je     0x22
  19:   b4 0e                   mov    $0xe,%ah
  1b:   bb 07 00                mov    $0x7,%bx
  1e:   cd 10                   int    $0x10
  20:   eb f2                   jmp    0x14
  22:   31 c0                   xor    %ax,%ax
  24:   cd 16                   int    $0x16
  26:   cd 19                   int    $0x19
  28:   ea f0 ff 00 f0          ljmp   $0xf000,$0xfff0
[...]
```

Commentaire.- Nous connaissons déjà l'option `-D` de l'utilitaire `objdump` servant au désassemblage. Par défaut, seuls les fichiers exécutables de format ELF sont désassemblés.

On utilise ici l'option `-b binary` pour les fichiers binaires (les fichiers `.com` de CP/M et de MS-DOS), c'est-à-dire ceux constitués uniquement de code machine, sans préfixe lié au système d'exploitation.

Il faut alors préciser à `objdump` la nature du code machine utilisé. On utilise pour cela l'option `-m` pour *Microprocesseur*. On a utilisé ici `i8086` pour le mode réel des microprocesseurs *Intel*. Il existe aussi `i386` pour le mode protégé et `x86-64` pour le mode 64 bits.

On aurait pu choisir `i386` mais cela aurait exiger des compléments d'information, introduits par l'option `-M` :

```
$ objdump -D -b binary -mi386 -M addr16,data8 ./sect1.com
```

en spécifiant que nous sommes dans le mode d'instructions 16 bits.

Par défaut, le désassemblage s'effectue en notation AT&T. On peut utiliser le paramètre `intel` (au lieu du paramètre par défaut `att`) pour `-M` pour désassembler en notation *Intel*.

14.2 L'amorçage

14.2.1 Pas de chargement direct du noyau Linux

Contexte.- Rappelons que le BIOS des PC copie le premier secteur du média de démarrage en mémoire vive à l'adresse physique `0x7C00` puis donne la main à cette adresse, de façon à exécuter le code qui vient d'être chargé. On dispose donc de 512 octets pour pouvoir faire quelque chose.

Dans ses premières versions (voir [Ceg-04], pp. 635–642), Linux disposait de son propre chargeur d'amorçage, écrit en langage d'assemblage, constituant la première moitié du fichier source `boot.s`.

Depuis l'apparition de l'utilitaire de multi-démarrage `LILO` (pour *LInux LOader*), on ne peut plus charger Linux directement : si on essaie de le faire, on obtient un message demandant d'utiliser un chargeur.

Manipulations.- Voyons comment ce dernier comportement est obtenu. Pour cela, récupérons le premier « secteur » du fichier noyau :

```
$ dd if=/boot/vmlinuz64-3.4.42 of=vmlinuz.dat count=1
```

et désassemblons-le :

```
$ objdump -D -b binary -mi8086 vmlinuz.dat
```

en comparant le résultat avec des extraits de `/usr/src/linux/arch/x86/boot/header.S`, le fichier source correspondant :

```
vmlinuz.dat:     file format binary

Disassembly of section .data:

00000000 <.data>:
                                                |39: .code16
                                                |40: .section ".bstext", "ax"
                                                |41:
                                                |42: .global bootsect_start
                                                |43: bootsect_start:
                                                |
                                                |50:    # Normalize the start address
   0:   ea 05 00 c0 07  ljmp   $0x7c0,$0x5      |51:    ljmp    $BOOTSEG, $start2
                                                |52:
                                                |53: start2:
   5:   8c c8           mov    %cs,%ax          |54:    movw    %cs, %ax
   7:   8e d8           mov    %ax,%ds          |55:    movw    %ax, %ds
   9:   8e c0           mov    %ax,%es          |56:    movw    %ax, %es
   b:   8e d0           mov    %ax,%ss          |57:    movw    %ax, %ss
   d:   31 e4           xor    %sp,%sp          |58:    xorw    %sp, %sp
   f:   fb              sti                     |59:    sti
  10:   fc              cld                     |60:    cld
                                                |61
  11:   be 2d 00        mov    $0x2d,%si        |62:    movw    $bugger_off_msg, %si
                                                |63
                                                |64 msg_loop:
  14:   ac              lods   %ds:(%si),%al    |65:    lodsb
  15:   20 c0           and    %al,%al          |66:    andb    %al, %al
  17:   74 09           je     0x22             |67:    jz      bs_die
  19:   b4 0e           mov    $0xe,%ah         |68:    movb    $0xe, %ah
  1b:   bb 07 00        mov    $0x7,%bx         |69:    movw    $7, %bx
  1e:   cd 10           int    $0x10            |70:    int     $0x10 /* print a character */
  20:   eb f2           jmp    0x14             |71:    jmp     msg_loop
                                                |72
                                                |73  bs_die:
                                                |74:     # Allow the user to press a key, then reboot
  22:   31 c0           xor    %ax,%ax          |75:     xorw    %ax, %ax
  24:   cd 16           int    $0x16            |76:     int     $0x16
```

```
 26:   cd 19           int    $0x19          |77      int     $0x19
                                             |78
                                             |79      # int 0x19 should never return.  In case it does anyway,
                                             |80      # invoke the BIOS reset code...
 28:   ea f0 ff 00 f0  ljmp   $0xf000,$0xfff0|81      ljmp    $0xf000,$0xfff0
                                             |82
                                             |
                                             |90
                                             |91      .section ".bsdata", "a"
                                             |92 bugger_off_msg:
 2d:   44 69 72 65 63 74 ....                |93      .ascii  "Direct booting is not supported. "
       ..............                        |94      .ascii  "Use a boot loader program instead.\r\n"
                                             |95      .ascii  "\n"
                                             |96      .ascii  "Remove disk and press any key to reboot ...\r\n"
 bc:   00                                    |97      .byte   0
```

Commentaires.- 1^o) Le fichier de code à charger (par le chargeur) pour le démarrage de Linux pour une architecture x86 est généré par le fichier `arch/x86/boot/header.S`. Celui-ci comprend 505 lignes dont le début du contenu, auquel nous avons ajouté la numérotation des lignes, est :

```
1   /*
2    * header.S
3    *
4    * Copyright (C) 1991, 1992 Linus Torvalds
5    *
6    * Based on bootsect.S and setup.S
7    * modified by more people than can be counted
8    *
9    * Rewritten as a common file by H. Peter Anvin (Apr 2007)
10   *
11   * BIG FAT NOTE: We're in real mode using 64k segments.  Therefore segment
12   * addresses must be multiplied by 16 to obtain their respective linear
13   * addresses. To avoid confusion, linear addresses are written using leading
14   * hex while segment addresses are written as segment:offset.
15   *
16   */
17
```

Ce fichier remplace les deux fichiers `bootsect.S` et `setup.S` écrits par Linus `Torvalds` lors de la première version de Linux, datant de 1991 (ligne 4), modifié depuis par un très grand nombre de personnes (ligne 7), pour tenir compte de différents cas, et fondus en un seul fichier en 2007 (ligne 9).

- 2^o) La ligne 11 rappelle, qu'au démarrage d'un microprocesseur *Intel*, on se trouve en mode réel.

- 3^o) Cette version de Linux est prévue, non seulement pour un microprocesseur *Intel* d'architecture IA-32 (dont IA-32e), mais plus précisément pour un PC : le code ne fonctionnera pas sur un Mac d'*Apple* bien que celui-ci utilise maintenant un tel microprocesseur.

```
28
29  BOOTSEG             = 0x07C0                /* original address of boot-sector */
```

Le BIOS d'un tel PC est en effet explicitement utilisé pour charger le secteur de démarrage à l'adresse `0x07C00` (ligne 29). Remarquez le sens ici, entre autres, de l'avertissement des lignes 11 à 13.

- 4^o) Rappelons que les mêmes instructions en langage d'assemblage se traduisent de façon différente en langage machine suivant que l'on se trouve en mode d'instructions 16 bits ou 32 bits : la signification des préfixes `0x66` et `0x67` n'est pas la même.

Il faut donc spécifier à l'assembleur que l'on démarre en mode réel et donc en mode d'instructions 16 bits, ce qui est fait à la ligne 39.

- 5^o) La première ligne de code, lorsque le fichier sera assemblé, commence après l'étiquette `bootsect_start`, ce que rappellent les lignes 42 et 43.

On peut oublier les lignes 44 à 48, prises en compte si on a l'option `CONFIG_EFI_STUB`, ce qui n'est pas notre cas.

La première instruction (`0x0` correspondant aux lignes 50 et 51 du fichier source) consiste en un saut long à la ligne suivante.

- 6^o) Classiquement, on initialise les registres de segment `DS`, `ES` et `SS` avec la valeur de `CS` (modèle `tiny`) (instructions d'adresses `0x5` à `0xb`, soit lignes 54 à 57), le pointeur de pile `SP` à zéro (instruction d'adresse `0xd`, soit ligne 58), on permet les interruptions masquables (instruction d'adresse `0xf`, soit ligne 59) et on progressera en avant lors des opérations sur les chaînes de caractères (instruction d'adresse `0x10`, soit ligne 60).

- 7^o) On affiche le message `bugger_off_msg` (instructions d'adresses `0x11` à `0x20`, soit lignes 62 à 71 et instructions d'adresses `0x2d` à `0xbc`, soit lignes 91 à 97) en utilisant l'interruption `0x10` du BIOS (affichage d'un caractère).

Bien entendu, que le premier secteur soit placé sur une disquette (comme indiqué), sur un disque dur ou une clé USB ne change rien.

Remarquons la façon dont est désassemblé le code `0xac` de l'instruction d'adresse `0x14`, là où *Intel* se contente de `lodsb`.

- 8^o) On passe ensuite (instruction d'adresse `0x17`, soit ligne 67) à l'instruction d'adresse `0x22`, soit ligne 73, pour redémarrer l'ordinateur (interruption `0x19` du BIOS) lorsqu'on frappera sur une touche (attente bloquante de l'interruption `0x16` du BIOS).

Par sécurité, si cela se passe mal, on revient à l'instruction exécutée lors du démarrage du microprocesseur (instruction d'adresse `0x28`, soit ligne 81), ce qui revient à un redémarrage (à chaud).

- 9^o) Le reste du fichier n'est donc pas utilisé dans ce cas. Il l'est lorsqu'on charge le fichier avec un utilitaire de multi-démarrage qui, lui, ne donne pas la main à la même ligne.

14.2.2 Utilitaire de démarrage multiple

Le BIOS d'origine du PC essaie de lire le premier secteur de la disquette du premier lecteur de disquette et passe à l'interpréteur BASIC s'il n'y a pas de disquette présente. Si une disquette est présente, il vérifie que ce premier secteur (de 512 octets) se termine par la signature `0x55`, `0xAA` et, s'il en est ainsi, copie ce secteur en mémoire vive à l'adresse `0x7C00` et lui donne la main. Si la signature n'est pas la bonne, un message demande de placer un autre disque.

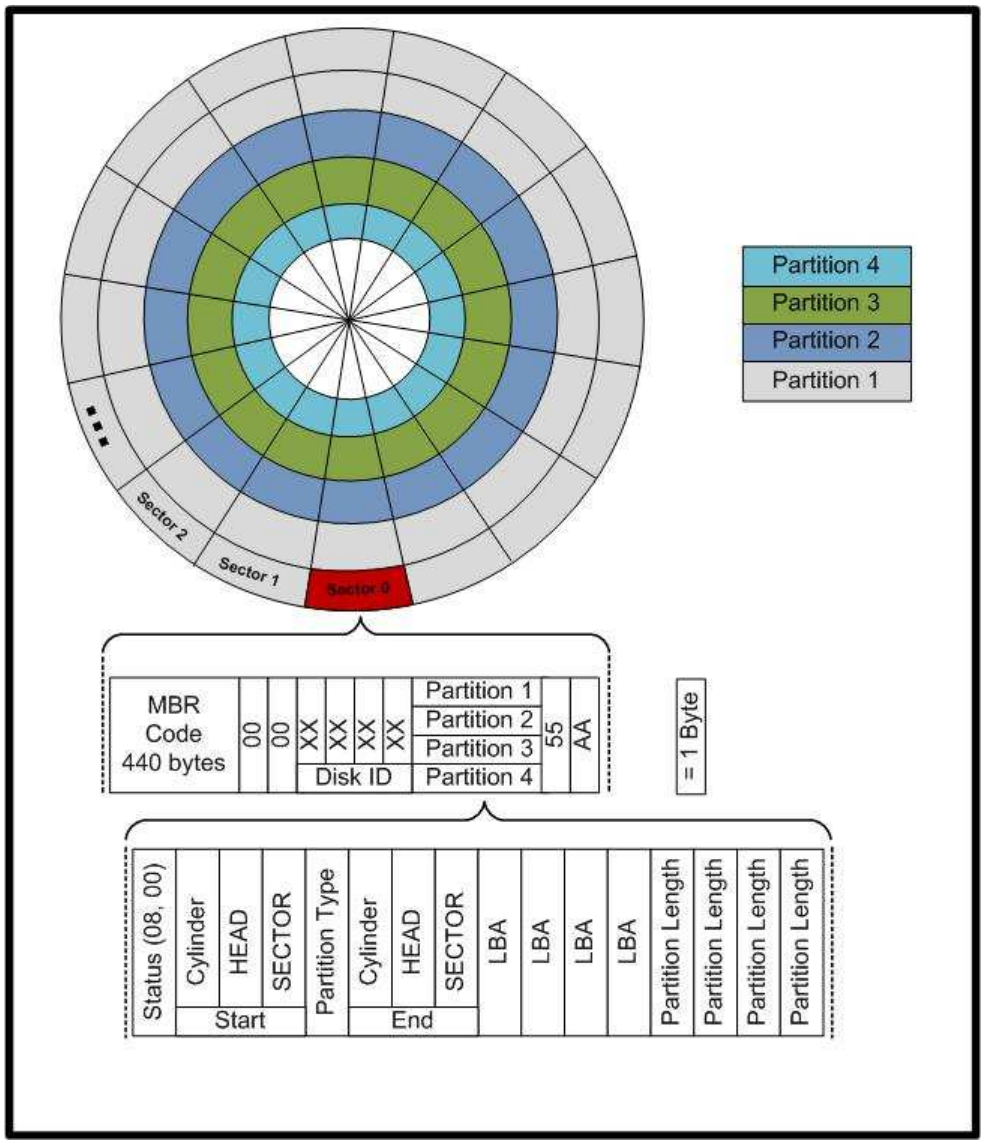

FIGURE 14.1 – Master Boot Record

Avec l'apparition des disques durs, le BIOS a changé. Il parcourt tous les médias rattachés à l'ordinateur, dans un ordre qui peut être choisi par l'utilisateur, en lit le premier secteur, le charge et l'exécute pour le premier média, dans l'ordre choisi, dont le premier secteur se termine par la signature `0x55`, `0xAA`.

Le premier secteur (de 512 octets) d'un disque, à savoir le secteur 1 du cylindre 0 de la tête 0, s'appelle **secteur de démarrage** ou **MBR** (pour *Master Boot Record*). Il est simulé sur une clé USB. La structure qu'en attend le BIOS est visualisée sur la figure 14.1 :

— Les 446 premiers octets constituent un programme.

— Les 64 octets suivants constituent la **table de partition**, contenant un enregistrement de 16 octets pour chacune des quatre **partitions** (dites **physiques**) possibles.

— Les deux derniers octets contiennent la signature `0x55`, `0xAA`.

Le programme situé sur les 446 premiers octets ne peut être qu'un **programme de pré-chargement** et affichant un message d'erreur si on ne parvient pas à pré-charger. Il ne peut être écrit qu'en mode réel. Le but de ce programme de pré-chargement est de trouver une **partition active**, de s'assurer que les trois autres partitions sont toutes inactives, de charger le **programme de chargement**, d'une taille en général supérieure à 512 octets, en mémoire vive et de lui donner la main.

Les deux programmes de pré-chargement et de chargement sont liés et constituent donc en général un même logiciel. Les plus célèbres, liés à Linux, sont `LILO`, dont nous avons déjà parlé (utilisant les secteurs physiques du disque dur), et **GRUB** (pour *GRand Unified Bootloader*), prenant en charge les systèmes de fichiers (`ext2` et `ext3`) de Linux.

Structure de l'image du noyau.- L'image du noyau de Linux est en partie compressée avec `zlib`, précédée d'une partie d'initialisation et de décompression de la partie compressée. Le rôle du chargeur est de trouver cette image, de la charger en mémoire vive et de lui donner la main.

14.2.3 Cas de LILO

Première étape : paramètres.- Si l'utilitaire de démarrage multiple de notre système est `LILO`, récupérons le premier secteur de notre premier disque dur :

```
$ sudo dd if=/dev/sda of=sw64boot.dat count=1
```

puis désassemblons-le, en comparant le résultat avec les sources, contenues dans le fichier des sources de `LILO`, et non de Linux, `lilo-23.2/src/first.S` :

```
$ objdump -D -b binary -m i8086 -Maddr16,data16 sw64boot.dat


sw64boot.dat:     file format binary


Disassembly of section .data:
                                        |60  .text
                                        |61
                                        |62  .globl  _main
                                        |63
                                        |64  .org    0
                                        |65
00000000 <.data>:                       |66  zero:
   0:   fa              cli             |67 _main:  cli                     ! NT 4 blows up if this is
                                        |                                   missing
   1:   eb 21           jmp    0x24     |68         jmp     start
                                        |69
   3:   01                              |70 stage:  .byte   STAGE_FIRST
                                        |71         .org    4
                                        |72 reloc:
                                        |73 #if RELOCATABLE
   4:   b4 01                           |74         .word   theend-zero     ! size of the code & params
                                        |75 #else
                                        |76         .word   0               ! no size indication
                                        |77 #endif
                                        |78         .org    6
                                        |79
                                        |80 ! Boot device parameters. They are set by the installer.
                                        |81
   6:   4c 49 4c 4f                     |82 sig:    .ascii  "LILO"
   a:   17 02                           |83 vers:   .word   VERSION
   c:   f4 3d d7 51                     |84 mapstamp: .long 0               ! map timestamp
                                        |85
                                        |86 length  =  *-sig                ! for the stage 1 vs stage 2
                                        |                                   comparison
                                        |87
  10:   00 00 00 00                     |88 raid:   .long   0               ! raid sector offset
  14:   f2 f4 7d 51                     |89 tstamp: .long   0               ! timestamp
  18:   34 15 50 38                     |90 map_serial_no: .long   0        ! volume S/N containing map file
  1c:   81 00                           |91 prompt: .word   0               ! indicates whether to always
                                        |                                   enter prompt
                                        |92                                 ! contains many other flags, too
                                        |93
  1e:   81                              |94 d_dev:  .byte   0x80            ! map file device code
  1f:   60                              |95 d_flag: .byte   0               ! disk addressing flags
  20:   96 74 07 10                     |96 d_addr: .long   0               ! disk addr of second stage
                                        |                                   index sector
                                        |
 1b3:   c3              ret             |567        ret
                                        |568 #endif
                                        |569
                                        |570 theend:
                                        |571
                                        |572 !
                                        |573 !   If 'first' loads as the MBR, then there must be space for
                                        |        the partition
                                        |574 !   table.  If 'first' loads as the boot record of some
                                        |        partition, then
                                        |575 !   the space reserved below is not used.  But we must
                                        |        reserve the area
                                        |576 !   as a hedge against the first case.
                                        |577 !
```

```
                                    |578 !
                                    |579         .org    MAX_BOOT_SIZE
                                    |580         .word   0,0,0,0         ! space for NT, DRDOS, and LiLO
                                    |                                      volume S/N
                                    |581
                                    |582         .org    0x1be           ! spot for the partition table
                                    |583 p_table:
                                    |584         .blkb   16
                                    |585         .blkb   16
                                    |586         .blkb   16
                                    |587         .blkb   16
                                    |588 #ifdef FIRST
                                    |589         .org    *-2
                                    |590         .long   FIRST           ! boot block check
                                    |591 #else
1fe:   55                           |592         .word   0xAA55          ! boot block signature
1ff:   aa                           |
```

Une première partie du fichier contient les paramètres dont `LILO` aura besoin. Puisqu'il n'y a pas de distinction entre segment de données et segment de code, la première instruction sera un saut passant ces données.

Effectivement, on ne permet pas les interruptions masquables (instruction d'adresse `0x0`, soit ligne 67, apparemment à cause d'un problème avec *Windows NT*) et on effectue un saut court à l'instruction d'adresse `0x24` grâce à l'instruction d'adresse `0x1`, soit ligne 68, ce qui nous laisse quelques lignes pour y placer des données.

Les paramètres sont :

— Un octet précisant qu'il s'agit de la première étape de `LILO`.
  La constante `STAGE_FIRST` est définie ligne 245 du fichier `lilo-23.2/src/lilo.h` :

```
#define STAGE_FIRST 1 /* first stage loader code */
```

— Un mot spécifiant la taille du code de la première partie de `LILO`, y compris les paramètres.
  On trouve ici `0x1b4`, soit 436, ce qui nous amène juste avant la table de partitions.

— Quatre caractères ASCII, "LILO", constituant la signature de `LILO`.

— Un mot double contenant une durée.

— Un mot double contenant éventuellement le décalage du secteur en cas d'un disque dur RAID, ce qui n'est pas le cas ici.

— Un mot double contenant une autre durée.

— Un mot pour le prompteur.

— Un octet spécifiant le code du disque dur contenant la seconde partie de `LILO`. On a ici `0x81` : il s'agit donc du second disque dur.

— Un octet spécifiant les indicateurs d'adressage de ce disque. Les valeurs en sont définies dans le fichier `lilo.h` :

```
266 #define LINEAR_FLAG     0x40    /* mark linear address */
267 #define LBA32_FLAG      0x20    /* mark lba 32-bit address */
268 #define LBA32_NOCOUNT   0x40    /* mark address with count absent */
269 #define RAID_REL_FLAG   0x10    /* mark address raid-relocatable */
270 /*
271  *  FLAG    Description
272  *
273  *  0x00    pure geometric addressing (C:H:S)
274  *  0x40    Linear address (24-bits) converted to CHS at boot-time
275  *  0x60    LBA32 address (32-bits), count=1, sets the high nibble!!
276  *  0x20    LBA32 address (24-bits) + (8-bit) high nibble (implied)
277  *
278 */
```

On a ici `0x60` pour spécifier un adressage LBA.

Un secteur est repéré sur un disque dur par une adresse, soit CHS, soit LBA.

L'**adresse CHS** (*Cylinder/Head/Sector*) spécifie les numéros de cylindre, de la tête et du secteur, comme son nom l'indique. La structure d'une donnée CHS a pour conséquence que les valeurs permises sont 0 à 1023 cylindres, 0 à 255 têtes et 1 à 63 secteurs. Si on multiplie ces valeurs, on voit que la capacité maximale d'un disque dur que l'on puisse gérer avec l'adressage CHS est de 8 GiO pour des secteurs de 512 octets.

Quand on est passé des disques IDE (de capacité maximale de 512 MiO) aux disques E-IDE (*enhanced IDE*), le standard a prévu le passage du mode d'adressage CHS à l'**adressage LBA** (*Logical Block Addressing*, adressage par bloc logique). En LBA, on n'a besoin de connaître que le nombre de secteurs depuis le début du disque jusqu'à la position du secteur recherché : les secteurs sont numérotés en commençant par 0.

Pour un disque possédant NT têtes par cylindre et NS secteurs par piste, la conversion d'une adresse LBA A vers CHS s'obtient de la façon suivante :

$$S = (A\%NS) + 1$$

$$H = (A - S + 1)/NS\%NT$$

$$C = (A - S + 1)/NS/NT$$

La conversion inverse d'une adresse CHS vers LBA est obtenue par :

$$A = (C \times NT \times NS) + (H \times NS) + S - 1$$

Deux versions d'adresses LBA existent pour les disques durs IDE : une première version utilise 28 bits pour coder l'adresse, ce qui permet de gérer des disques d'une capacité allant jusqu'à 128 GiO ; la seconde version (définie en 2002 dans la norme ATA/ATAPI-6) utilise 48 bits, ce qui permet de gérer des disques d'une capacité allant jusqu'à 128 PiO.

— Un mot double spécifiant l'adresse LBA du secteur de la fin de la seconde partie de `LILO`.

On a ici le secteur `0x10077496`, soit 268 924 054.

Deuxième étape : initialisations.- Comme d'habitude, le désassembleur est pertubé par les données. Continuons donc en lui spécifiant explicitement l'adresse à partir de laquelle il faut désassembler, à savoir `0x24`, comme nous l'avons vu :

```
$ objdump -D -b binary -m i8086 -Maddr16,data16 --start-address=0x24 sw64boot.dat
```

```
                                      |124 /*****************************************/
                                      |125 !  The following instruction MUST be
                                      |126 !  first instruction after the CLI/JMP short
                                      |127 !  at the start of the file; otherwise
                                      |128 !  the boot sector relocation fails.
                                      |129 !
                                      |130 start:
  24:   b8 c0 07        mov    $0x7c0,%ax      |131 mov  ax,#BOOTSEG   ! use DS,ES,SS = 0x07C0
                                      |132 /*****************************************/
                                      |133
  27:   8e d0           mov    %ax,%ss         |134 mov  ss,ax
  29:   bc 00 08        mov    $0x800,%sp      |135 mov  sp,#SETUP_STACKSIZE ! set the stack for First Stage
  2c:   fb              sti                    |136 sti                      ! now it is safe
                                      |137
  2d:   52              push   %dx             |138 push dx                  ! set ext_dl (and ext_dh, which
                                      |                                       is not
                                      |                                       used)
  2e:   53              push   %bx             |139 push bx                  ! WATCH the order of pushes
  2f:   06              push   %es             |140 push es                  ! set ext_es
  30:   56              push   %si             |141 push si                  ! set ext_si
                                      |
  31:   fc              cld                    |150 cld                      ! do not forget to do this !!!
  32:   8e d8           mov    %ax,%ds         |151 mov ds,ax                ! address data area
  34:   31 ed           xor    %bp,%bp         |152 xor bp,bp                ! shorted addressing
                                      |153
```

— On initialise la pile pour la première étape, c'est-à-dire celle de préchargement (instructions d'adresses `0x24` à `0x29`, soit lignes 124 à 135).

Les constantes `BOOTSEG` et `SETUP_STACKSIZE` sont définies dans le fichier `lilo.h` :

```
376 #ifdef LILO_ASM
377 BOOTSEG   = 0x07C0                  ! original address of boot-sector
[...]
398 SETUP_STACKSIZE = 2048              ! stacksize for kernel setup.S
```

L'adresse de segment `0x07C0` est celle à laquelle le BIOS a chargé le programme de pré-chargement, dont on n'a plus besoin. La taille de la pile est de 2 048 octets.

— On peut alors rétablir les interruptions (instruction d'adresse `0x2c`, soit ligne 136).

— On sauvegarde le contenu des registres (instructions d'adresses `0x2d` à `0x30`, soit lignes 138 à 141) et on progressera en avant lors des opérations sur les chaînes de caractères (instruction `0x31`, soit ligne 150).

— Le segment des données est confondu avec le segment de code (instruction `0x32`, soit ligne 151) et on initialise le registre BP à 0 (instruction `0x34`, soit ligne 152).

Troisième étape : affichage de 'L'.- `LILO` affiche la lettre 'L' lors de la première étape :

```
                                        |40  ! VIDEO_ENABLE   for those systems that disable the video
                                        |                     on boot
                                        |41  !   = 0          first stage does not enable video
                                        |42  !   = 1          use get vid mode/set vid mode to enable
                                        |43  !   = 2          use VGA enable call to enable video
                                        |44  !                (cannot use, as code gets too big)
                                        |45  !   = 3          use direct mode set (mode 3, CGA, EGA, VGA)
                                        |46  !   = 7          use direct mode set (mode 7, MDA)
                                        |47  !
                                        |48  #ifndef VIDEO_ENABLE
                                        |49  # if VALIDATE==0
                                        |50  #  define VIDEO_ENABLE 2
                                        |51  # else
                                        |52  #  define VIDEO_ENABLE 2
                                        |53  # endif
                                        |54  #endif
                                        |
                                        |154 #if VIDEO_ENABLE
                                        |155 ! a BIOS has been found where the video interrupt (0x10)
                                        |       trashes DX
                                        |156 !   so, we had better be very paranoid about DX
                                        |157 !
                                        |158 # if VIDEO_ENABLE==2
 36:   60              pusha             |159 pusha                    ! protect DX
                                        |160 # endif
                                        |
                                        |167 # else  /* VIDEO_ENABLE==2 */
 37:   b8 00 12        mov    $0x1200,%ax|168 mov ax,#0x1200           ! enable video (VGA)
 3a:   b3 36           mov    $0x36,%bl  |169 mov bl,#0x36             ! (probably a nop on EGA or MDA)
                                        |170 # endif
 3c:   cd 10           int    $0x10      |171 int 0x10                ! video call
                                        |172 # if VIDEO_ENABLE==2
 3e:   61              popa              |173 popa                     ! restore DX
                                        |174 # endif
                                        |175 #endif
                                        |176
                                        |177 #if (VIDEO_ENABLE&1) == 0
 3f:   b0 0d           mov    $0xd,%al   |178 mov al,#0x0d             ! gimme a CR ...
 41:   e8 66 01        call   0x1aa      |179 call display
                                        |180 ; the suspect call for trashing DX on one BIOS:
 44:   b0 0a           mov    $0xa,%al   |181 mov al,#0x0a             ! ... an LF ...
 46:   e8 61 01        call   0x1aa      |182 call display
                                        |183 #endif
                                        |184
                                        |
 49:   b0 4c           mov    $0x4c,%al  |189 mov al,#0x4c             ! ... an 'L' ...
 4b:   e8 5c 01        call   0x1aa      |190 call display
                                        |191
                                        |
                                        |534 display0:
                                        |535 #ifndef LCF_NOVGA
                                        |536 display:
                                        |537 #endif
1aa:   60              pusha             |538         pusha            ! make sure no register is changed
1ab:   bb 07 00        mov    $0x7,%bx   |539         mov    bx,#7     ! BH=0, BL=07
1ae:   b4 0e           mov    $0xe,%ah   |540         mov    ah,#14
1b0:   cd 10           int    $0x10      |541         int    0x10
1b2:   61              popa              |542         popa             ! restore all the registers
                                        |543 #ifdef LCF_NOVGA
                                        |544 display:
                                        |545 #endif
1b3:   c3              ret               |546         ret
                                        |547
```

— On spécifie que le contenu de la mémoire graphique doit être affiché (instructions d'adresses `0x36` à `0x3e`, soit lignes 154 à 176), en utilisant l'interruption `0x10` du BIOS, fonction `0x12`, sous-fonctions `0x36` et paramètre `0x0`.
— On effectue un passage à la ligne (instructions d'adresses `0x3f` à `0x46`, soit lignes 178 à 182) : le code ASCII du retour chariot est `0x0d`, celui du passage à la ligne suivante est `0x0a`.

La sous-routine `display` est définie par les instructions d'adresses `0x1aa` à `0c1b3`, soit lignes 538 à 546.

Remarquons la présence de 'gimme' dans le commentaire, contraction familière de 'give me'.
— On affiche ensuite 'L' (instructions d'adresses `0x49` et `0x4b`, soit lignes 189 et 190).

Quatrième étape : tenir compte de la commande de démarrage éventuelle.- L'étape suivante permet de tenir compte de la commande de démarrage éventuelle :

```
                                        |192 lagain:
4e:   60             pusha              |193 pusha                    ! preserve all the registers
                                        |                             !  for restart
                                        |194
4f:   1e             push   %ds         |195 push ds
50:   07             pop    %es         |196 pop  es                  ! use buffer at end of boot sector
                                        |197
51:   80 fa fe       cmp    $0xfe,%dl   |198 cmp dl,#EX_DL_MAG        ! possible boot command line
                                        |                             !  (chain.S)
54:   75 02          jne    0x58        |199 jne boot_in_dl
56:   88 f2          mov    %dh,%dl     |200 mov dl,dh                ! code passed in DH instead
                                        |201 boot_in_dl:
                                        |202
```

— On sauvegarde le contenu de tous les registres (instruction d'adresse `0x4e`, soit ligne 193), on confond le segment supplémentaire avec le segment de code, modèle *tiny* oblige, et on regarde si le registre `DX` contient une commande de démarrage (instruction `0x51`, soit ligne 198).

La constante `EX_DL_MAG` est définie dans le fichier `lilo.h` :

```
283 #define EX_DL_MAG 0xfe /* magic number in DL */
```

— S'il en est ainsi, on place cette commande dans le registre `DH` (instruction d'adresse `0x56`, soit ligne 200).

Cinquième étape : recherche du disque contenant la seconde partie de LILO.- On cherche ensuite le disque contenant la seconde partie de `LILO` :

```
58:   bb 00 02        mov    $0x200,%bx       |203    mov    bx,#map2       ! buffer for volume search
5b:   8a 76 1e        mov    0x1e(%bp),%dh    |204    mov    dh,[d_dev](bp) ! map device to DH
                                              |205
                                              |206 #if VALIDATE
5e:   89 d0           mov    %dx,%ax          |207    mov    ax,dx          ! copy device code to AL
60:   80 e4 80        and    $0x80,%ah        |208    and    ah,#0x80       ! AH = 00 or 80
63:   30 e0           xor    %ah,%al          |209    xor    al,ah          ! hi-bits must be the same
65:   78 0a           js     0x71             |210    js     use_installed
67:   3c 10           cmp    $0x10,%al        |211    cmp    al,#MAX_BIOS_DEVICES ! limit the device code
69:   73 06           jae    0x71             |212    jae    use_installed  ! jump if DL is not valid
                                              |213 #endif
                                              |214
                                              |215 ! map is on boot device for RAID1, and if so marked; viz.,
                                              |216
6b:   f6 46 1c 40     testb  $0x40,0x1c(%bp)  |217    test   byte ptr [prompt](bp),#FLAG_MAP_ON_BOOT
6f:   75 2e           jne    0x9f             |218    jnz    use_boot       ! as passed in from BIOS or MBR
                                              |                               loader
                                              |219
                                              |220 use_installed:
71:   88 f2           mov    %dh,%dl          |221    mov    dl,dh          ! device code to DL
73:   66 8b 76 18     mov    0x18(%bp),%esi   |222    mov    esi,[map_serial_no](bp) ! to search for
77:   66 09 f6        or     %esi,%esi        |223    or     esi,esi
7a:   74 23           je     0x9f             |224    jz     done_search
                                              |225
7c:   52              push   %dx              |226    push   dx             ! save flags
                                              |227
7d:   b4 08           mov    $0x8,%ah         |228    mov    ah,#8          ! get number of hard disks
7f:   b2 80           mov    $0x80,%dl        |229    mov    dl,#0x80
81:   53              push   %bx              |230    push   bx             ! paranoia
82:   cd 13           int    $0x13            |231    int    0x13
84:   5b              pop    %bx              |232    pop    bx
85:   72 57           jb     0xde             |233    jc     error
                                              |234
87:   0f b6 ca        movzbw %dl,%cx          |235    movzx  cx,dl          ! extend to word in CX
                                              |236
                                              |237 #if GEOMETRIC
8a:   ba 7f 00        mov    $0x7f,%dx        |238    mov    dx,#0x80-1     ! device 80, flags=0
                                              |
                                              |243 vagain:
8d:   42              inc    %dx              |244    inc    dx
8e:   66 31 c0        xor    %eax,%eax        |245    xor    eax,eax
                                              |246 #if GEOMETRIC
91:   40              inc    %ax              |247    inc    ax             ! geometric addressing
                                              |248 #endif
92:   e8 60 00        call   0xf5             |249    call   disk_read      ! read
                                              |250
95:   66 3b b7 b8 01  cmp    0x1b8(%bx),%esi  |251    cmp    esi,[PART_TABLE_OFFSET-6](bx)
9a:   74 03           je     0x9f             |252    je     vol_found
9c:   e2 ef           loop   0x8d             |253    loop   vagain
                                              |254
9e:   5a              pop    %dx              |255    pop    dx             ! restore specified BIOS code
                                              |256                           ! AX and DX are identical at
                                              |                               this point
                                              |257
                                              |
                                              |595 ! Better be exactly 0x200
                                              |596 map2 equ *                ! addressed as ES:[map2]
```

— On fait pointer le registre `BX` sur la fin du secteur (instruction d'adresse `0x58`, soit ligne 203) : la variable `map2` est définie à la fin du fichier source (lignes 595 et 596). On place le numéro de volume placé en paramètre (ligne 94) dans le registre `DH` (instruction d'adresse `0x5b`, soit ligne 204) : rappelons que `BP` a été initialisé à 0. On place ce numéro de volume dans le registre `AL` (instruction d'adresse `0x5e`, soit ligne 207). On vérifie que ce volume est présent (instructions d'adresses `0x60` à `0x69`, soit lignes 208 à 212). Si ce n'est pas le cas, c'est qu'on utilise RAID1 (instructions d'adresses `0x6b` et `0x6f`, soit lignes 215 à 218).

— On place le numéro de volume dans le registre `DL` (instruction d'adresse `0x71`, soit ligne 221). On place le numéro de série de ce volume (l'un des paramètres, celui de la ligne 90) dans le registre `ESI` (instruction d'adresse `0x73`, soit ligne 222). Si ce numéro de série est nul (instruction d'adresse `0x77`, soit ligne 223), on n'a pas à vérifier qu'il s'agit bien du disque portant ce numéro de série ; on a donc terminé la recherche (instruction d'adresse `0x7a`, soit ligne 224).

— Sinon, on lit les paramètres du disque. Rappelons la sémantique de la fonction `0x8` de l'interruption `0x13` du BIOS :

```
INT 0x13 AH = 0x8: Read Drive Parameters
   Parameters:
      AH = 0x8
      DL = drive index (e.g. 1st HDD = 0x80)
      ES:DI set to 0x0000:0x0000 to work around some buggy BIOS
   Results:
      CF Set On Error, Clear If No Error
      AH Return Code
      DL number of hard disk drives
      DH[4] logical last index of heads
            = number - 1 (because index starts with 0)
      CX [7:6] [15:8][4] logical last index of cylinders = number - 1
      CX [5:0][4] logical last index of sectors per track
            = number (because index starts with 1)
      BL[4] drive type (only AT/PS2 floppies)
      ES:DI[4] pointer to drive parameter table (only for floppies)
```

— On sauvegarde les indicateurs du volume sur la pile (instruction d'adresse `0x7c`, soit ligne 226) et on récupère les paramètres du disque dur (instructions d'adresses `0x7d` à `0x84`, soit lignes 228 à 232), grâce à l'interruption `0x13` du BIOS. Il y a une erreur si on ne parvient pas à les récupérer (instruction `0x84`, soit ligne 233).

— Le nombre de disques durs, tel qu'il est récupéré, est placé dans le registre `CX` (instruction d'adresse `0x87`, soit ligne 235) et on parcourt les disques durs (instructions d'adresses `0x8a` et `0x8d`, soit lignes 237 à 244). On initialise le registre `EAX` à 1 (instructions d'adresses `0x8e` et `0x91`, soit lignes 245 à 247) et on lit la valeur sur le disque (instruction d'adresse `0x92`, soit ligne 249).

— Une fois qu'on l'a lu, si la valeur est égale au numéro de série, on a trouvé le volume (instructions `0x95` et `0x9a`, soient lignes 251 et 252).

La constante `PART_TABLE_OFFSET` est définie dans le fichier `lilo.h` :

```
189 #define PART_TABLE_OFFSET  0x1be /* offset in the master boot sector */
```

— Sinon on passe au volume suivant (instruction `0x9c`, soit ligne 253) jusqu'à ce qu'on trouve.

Routine de lecture d'un paquet sur le disque.- La routine de lecture d'un « paquet » (de 16 octets) est définie à partir de l'instruction `0xf5`, soit ligne 355 :

```
                                        |355 ! packet read routine
                                        |356
                                        |357 disk_read:
                                        |
 f5:   60              pusha             |361    pusha
                                        |
 f6:   55              push   %bp        |368    push   bp              ! BP==0
 f7:   55              push   %bp        |369    push   bp              ! BP==0
                                        |370
 f8:   66 50           push   %eax       |371                            ! low order disk address
                                        |
 fa:   06              push   %es        |378    push   es              ! memory segment ES
 fb:   53              push   %bx        |379    push   bx              ! memory offset BX
 fc:   6a 01           push   $0x1       |380    push   #1              ! sector count
 fe:   6a 10           push   $0x10      |381    push   #16             ! size of packet = 16 bytes
100:   89 e6           mov    %sp,%si    |382    mov    si,sp           ! address of packet  DS:SI
                                        |383
102:   53              push   %bx        |384    push   bx
                                        |385
103:   f6 c6 60        test   $0x60,%dh  |386    test   dh,#LINEAR_FLAG|LBA32_FLAG
106:   74 70           je     0x178      |387    jz     disk_geometric
                                        |388
108:   f6 c6 20        test   $0x20,%dh  |389    test   dh,#LBA32_FLAG
10b:   74 14           je     0x121      |390    jz     disk_convert   ; it must be LINEAR
                                        |391
10d:   bb aa 55        mov    $0x55aa,%bx|392    mov    bx,#0x55AA     ;magic number
110:   b4 41           mov    $0x41,%ah  |393    mov    ah,#0x41
112:   cd 13           int    $0x13      |394    int    0x13
114:   72 0b           jb     0x121      |395    jc     disk_convert
116:   81 fb 55 aa     cmp    $0xaa55,%bx|396    cmp    bx,#0xAA55     ;changed?
11a:   75 05           jne    0x121      |397    jne    disk_convert
11c:   f6 c1 01        test   $0x1,%cl   |398    test   cl,#EDD_PACKET ;EDD packet calls supported
11f:   75 41           jne    0x162      |399    jnz    disk_edd
                                        |400
                                        |401  disk_convert:
121:   52              push   %dx        |402    push   dx
122:   06              push   %es        |403    push   es              ! protect on floppies
123:   b4 08           mov    $0x8,%ah   |404    mov    ah,#8           ! get geometry
125:   cd 13           int    $0x13      |405    int    0x13
127:   07              pop    %es        |406    pop    es
                                        |407  disk_error3:              ! transfer through on CF=1
128:   72 b4           jb     0xde       |408    jc     error           ! disk_error12
                                        |409
                                        |410 #if !CYL1023
12a:   51              push   %cx        |411    push   cx
12b:   c0 e9 06        shr    $0x6,%cl   |412    shr    cl,#6           ;;;;
12e:   86 e9           xchg   %ch,%cl    |413    xchg   cl,ch           ;CX is max cylinder number
130:   89 cf           mov    %cx,%di    |414    mov    di,cx           ;DI saves it
132:   59              pop    %cx        |415    pop    cx
                                        |416 #endif
133:   c1 ea 08        shr    $0x8,%dx   |417    shr    dx,#8
136:   92              xchg   %ax,%dx    |418    xchg   ax,dx           ;AX <- DX
137:   40              inc    %ax        |419    inc    ax              ;AX is number of heads
                                        |                               (256 allowed)
                                        |420
                                        |421  ; compensate for Davide BIOS bug
138:   49              dec    %cx        |422    dec    cx              ; 1..63 -> 0..62;  0->63
139:   83 e1 3f        and    $0x3f,%cx  |423    and    cx,#0x003f      ;CX is number of sectors
13c:   41              inc    %cx        |424    inc    cx              ; allow Sectors==0 to mean 64
                                        |425
13d:   f7 e1           mul    %cx        |426    mul    cx              ; kills DX also
13f:   93              xchg   %ax,%bx    |427    xchg   ax,bx           ;save in BX
                                        |428
140:   8b 44 08        mov    0x8(%si),%ax|429    mov    ax,[edd_d_addr](si)     ;low part of address
143:   8b 54 0a        mov    0xa(%si),%dx|430    mov    dx,[edd_d_addr+2](si)   ;hi part of address
                                        |431
146:   39 da           cmp    %bx,%dx    |432    cmp    dx,bx
148:   73 92           jae    0xdc       |433    jae    disk_error2    ;prevent division error
                                        |434
14a:   f7 f3           div    %bx        |435    div    bx              ;AX is cyl, DX is head/sect
                                        |436 #if CYL1023
```

```
                                                |437    cmp    ax,#1023
                                                |438 #else
14c:   39 f8           cmp    %di,%ax           |439    cmp    ax,di
                                                |440 #endif
14e:   77 8c           ja     0xdc              |441    ja     disk_error2    ;cyl is too big
                                                |442
150:   c0 e4 06        shl    $0x6,%ah          |443    shl    ah,#6          ; save hi 2 bits
153:   86 e0           xchg   %ah,%al           |444    xchg   al,ah
155:   92              xchg   %ax,%dx           |445    xchg   ax,dx
156:   f6 f1           div    %cl               |446    div    cl             ;AH = sec-1, AL = head
158:   08 e2           or     %ah,%dl           |447    or     dl,ah          ;form Cyl/Sec
15a:   89 d1           mov    %dx,%cx           |448    mov    cx,dx
15c:   41              inc    %cx               |449    inc    cx             ;sector is 1 based
                                                |450
15d:   5a              pop    %dx               |451    pop    dx             ! restore device code
15e:   88 c6           mov    %al,%dh           |452    mov    dh,al          ! set head#
160:   eb 1c           jmp    0x17e             |453    jmp    disk_read2
                                                |454
                                                |455
                                                |456
                                                |457 disk_edd:
162:   b4 42           mov    $0x42,%ah         |458    mov    ah,#0x42
                                                |459 disk_int13:
164:   5b              pop    %bx               |460    pop    bx
                                                |461
165:   bd 05 00        mov    $0x5,%bp          |462    mov    bp,#5
                                                |463 disk_retry:
168:   60              pusha                    |464    pusha
169:   cd 13           int    $0x13             |465    int    0x13
                                                |
16b:   73 16           jae    0x183             |470    jnc    disk_okay
                                                |471
16d:   4d              dec    %bp               |472    dec    bp             ! does not alter CF, already 0
16e:   74 b8           je     0x128             |473    jz     disk_error3    ! go to "jc" with CF=1 & ZF=1
                                                |474
170:   31 c0           xor    %ax,%ax           |475    xor    ax,ax          ! reset the disk controller
172:   cd 13           int    $0x13             |476    int    0x13
174:   61              popa                     |477    popa                  ! reset AX,BX,CX,DX,SI
175:   4d              dec    %bp               |478    dec    bp             ! fix up BP count
176:   eb f0           jmp    0x168             |479    jmp    disk_retry
                                                |480
                                                |481
                                                |482 disk_geometric:
178:   66 50           push   %eax              |483    push   eax
17a:   59              pop    %cx               |484    pop    cx
17b:   58              pop    %ax               |485    pop    ax
17c:   88 e6           mov    %ah,%dh           |486    mov    dh,ah
                                                |487
                                                |488 disk_read2:
17e:   b8 01 02        mov    $0x201,%ax        |489    mov    ax,#0x201      ;read, count of 1
181:   eb e1           jmp    0x164             |490    jmp    disk_int13
                                                |491
                                                |492
                                                |493 disk_okay:
                                                |494 ; the pusha block is implicitly removed below
                                                |495 ;;; mov (si+2*16-1),ah ! set error code
                                                |496 ;   the error code is never checked
183:   8d 64 10        lea    0x10(%si),%sp     |497     lea   sp,(si+16)     ! do not touch carry;
186:   61              popa                     |498     popa
                                                |
187:   c3              ret                      |502     ret
```

— On sauvegarde sur la pile tous les registres (instruction d'adresse `0xf5`, soit ligne 361), deux fois le contenu du registre `BP`, c'est-à-dire 0 (instructions d'adresses `0xf6` et `0xf7`, soit lignes 368 et 369), l'adresse du segment supplémentaire (instruction d'adresse `0xfa`, soit ligne 378), le décalage dans la mémoire (instruction d'adresse `0xfb`, soit ligne 379), le nombre de secteurs à lire, soit 1 (instruction d'adresse `0xfc`, soit ligne 380), la taille d'un paquet, soit 16 octets (instruction d'adresse `0xfe`, soit ligne 381) et une seconde fois le décalage en mémoire (instruction d'adresse `0x102`, soit ligne 384). On place l'adresse du paquet dans le registre `SI` (instruction d'adresse `0x100`, soit ligne 382).

— Si les paramètres du disque spécifient qu'on a affaire à un adressage LBA (instructions d'adresses 0x103 à 0x10b, soit lignes 386 à 390), ce qui est notre cas comme nous l'avons vu lors de l'étude des paramètres, on vérifie que le disque accepte les fonctions étendues du BIOS.

Rappelons les fonctionnalités de la fonction 0x41 de l'interruption 0x13 du BIOS :

```
INT 0x13 AH = 0x 41: Check Extensions Present
   Parameters:
      AH = 0x41
      DL = drive index (e.g. 1st HDD = 0x80)
      BX = 0x55AA
Results:
      CF Set On Not Present, Clear If Present
      AH Error Code or Major Version Number
      BX AA55h
      CX Interface support bitmask:
         1 - Device Access using the packet structure
         2 - Drive Locking and Ejecting
         4 - Enhanced Disk Drive Support (EDD)
```

— On place donc le nombre magique 0x55AA dans le registre BX (instruction d'adresse 0x10d, soit ligne 392), la valeur 0x41 dans le registre AH (instruction d'adresse 0x110, soit ligne 393) et on appelle l'interruption 0x13 (instruction d'adresse 0x112, soit ligne 394). Si l'indicateur CF est positionné ou si le contenu de BX a changé, on n'est pas en présence de LBA, ce qui n'est pas notre cas : il faut alors effectuer une conversion (instructions d'adresses 0x114 à 0x11a, soit lignes 395 à 397). Si EDD est supporté, on utilise cette possibilité (instructions d'adresses 0x11c et 0x11f, soit lignes 398 et 399).

— Puisqu'il y a toutes les chances qu'il en soit ainsi avec votre ordinateur. Passons donc rapidement sur l'adressage CHS (instructions d'adresses 0x121 à 0x160, soit lignes 401 à 453).

Avant d'aborder le cas de l'adressage LBA, rappelons les fonctionnalités de la fonction 0x42 de l'interruption 0x13 du BIOS :

```
INT 0x13 AH = 0x42: Extended Read Sectors From Drive
   Parameters:
      AH    = 0x42
      DL    = drive index (e.g. 1st HDD = 0x80)
      DS:SI = segment:offset pointer to the DAP
```

DAP (Disk Address Packet)		
offset	range size	description
00h	1 byte	size of DAP = 16 = 10h
01h	1 byte	unused, should be zero
02h..03h	2 bytes	number of sectors to be read
04h..07h	4 bytes	segment:offset pointer to the memory buffer to which sectors will be transferred (note that x86 is little-endian: if declaring the segment and offset separately, the offset must be declared before the segment
08h..0Fh	8 bytes	absolute number of the start of the sectors to be read (1st sector of drive has number 0)

```
Results:
   CF Set On Error, Clear If No Error
   AH Return Code
```

— Dans le cas où on peut utiliser EDD, on place le numéro de fonction `0x42` dans le registre `AH` (instruction d'adresse `0x162`, soit ligne 458), on récupère le décalage en mémoire dans le registre `BX` (instruction d'adresse `0x164`, soit ligne 460), on place 5 dans le registre `BP` pour passer les six valeurs de la pile ne nous intéressant pas dans ce cas (instruction d'adresse `0x165`, soit ligne 462), on restaure les valeurs des registres (instruction d'adresse `0x168`, soit ligne 464) et on appelle l'interruption `0x13` (instruction d'adresse `0x169`, soit ligne 465).

— S'il y a une erreur au cours de cette interruption (instruction d'adresse `0x16b`, soit ligne 470), on essaie à nouveau (instructions d'adresses `0x16d` à `0x181`, soit lignes 472 à 490). Sinon on incrémente l'index de 16 (instruction d'adresse `0x183`, soit ligne 497) et on récupère la valeur des registres (instruction d'adresse `0x186`, soit ligne 498).

Sixième étape : copie de la seconde partie de LILO en mémoire.- Une fois trouvé le volume contenant la seconde partie de `LILO`, on copie cette seconde partie en mémoire vive :

```
                                        |258 vol_found:
                                        |259                          ! uses value in DX, stack may
                                        |                              have extra value
                                        |260
                                        |261 done_search:
                                        |262 use_boot:
9f:   53              push   %bx        |263     push bx              ! save map2 for later
                                        |264
a0:   8a 76 1f        mov    0x1f(%bp),%dh |265     mov  dh,[d_flag](bp) ! get device flags to DH
a3:   be 20 00        mov    $0x20,%si  |266     mov  si,#d_addr
a6:   e8 df 00        call   0x188      |267     call pread           ! increments BX
                                        |268
a9:   b4 99           mov    $0x99,%ah  |269     mov  ah,#0x99        ! possible error code
ab:   66 81 7f fc 4c 49 4c 4f    cmpl   $0x4f4c494c,-0x4(%bx) |270 cmp  dword (bx-4),#EX_MAG_HL ! "LILO"
b3:   75 29           jne    0xde       |271     jne  error
                                        |272
b5:   5e              pop    %si        |273     pop  si              ! point at #map2
                                        |274
                                        |275 #if  1
b6:   68 80 08        push   $0x880     |276     push #SETUP_STACKSIZE/16 + BOOTSEG + SECTOR_SIZE/16*2
b9:   07              pop    %es        |277     pop  es
                                        |278 #else
                                        |
                                        |282 #endif
ba:   31 db           xor    %bx,%bx    |283     xor  bx,bx
                                        |284
                                        |285 sload:
bc:   e8 c9 00        call   0x188      |286     call pread           ! read using map at DS:SI
bf:   75 fb           jne    0xbc       |287     jnz  sload           ! into memory at ES:BX (auto
                                        |                              increment)
                                        |288
                                        |
                                        |315
                                        |316
                                        |317
                                        |318
                                        |319 disk_error2:
dc:   b4 40           mov    $0x40,%ah  |320     mov  ah,#0x40        ; signal seek error
                                        |321
                                        |322 ! no return from error
                                        |323 error:
                                        |324
                                        |325 #ifndef LCF_NO1STDIAG
de:   b0 20           mov    $0x20,%al  |326     mov  al,#32          ! display a space
e0:   e8 c7 00        call   0x1aa      |327     call display0
                                        |328
e3:   e8 b4 00        call   0x19a      |329     call bout
                                        |330 #endif
                                        |331
                                        |332 #ifndef DEBUG_LARGE
e6:   fe 4e 00        decb   0x0(%bp)   |333     dec  byte [zero](bp) !  CLI == 0xFA == 250
e9:   74 07           je     0xf2       |334     jz   zzz
                                        |335
                                        |336 #ifndef DEBUG_NEW
eb:   bc e8 07        mov    $0x7e8,%sp |337     mov  sp,#SETUP_STACKSIZE-4*2-8*2  ! set the stack
                                        |                                          for First Stage
                                        |338 #else
                                        |
                                        |340 #endif
ee:   61              popa               |341     popa                 ! restore registers for restart
ef:   e9 5c ff        jmp    0x4e       |342     jmp  near lagain     ! redo from start
                                        |343 #endif
                                        |344
                                        |345
                                        |346 zzz:
                                        |347 #ifndef DEBUG_NEW
f2:   f4              hlt                |348     hlt
                                        |349 #endif
f3:   eb fd           jmp    0xf2       |350     jmp  zzz             ! spin; wait for Ctrl-Alt-Del
                                        |
                                        |506 ! Pointer Read -- read using pointer in DS:SI
```

```
                                                |507
                                                |508 pread:
                                                |
188:   66 ad           lods   %ds:(%si),%eax    |509     lodsd                ! get address
18a:   66 09 c0        or     %eax,%eax         |510     or   eax,eax
18d:   74 0a           je     0x199             |511     jz   done
18f:   66 03 46 10     add    0x10(%bp),%eax    |512     add  eax,[raid](bp)  ! reloc is 0 on non-raid
193:   e8 5f ff        call   0xf5              |513     call disk_read
                                                |514
196:   80 c7 02        add    $0x2,%bh          |515     add  bh,#SECTOR_SIZE/256  ! next sector
                                                |516 done:
199:   c3              ret                      |517     ret
                                                |518
                                                |519
                                                |520
                                                |521
                                                |522 #if !defined(LCF_NO1STDIAG) || defined(DEBUG_NEW)
19a:   c1 c0 04        rol    $0x4,%ax          |523 bout: rol  ax,#4          ! bring hi-nibble to position
19d:   e8 03 00        call   0x1a3             |524     call nout
1a0:   c1 c0 04        rol    $0x4,%ax          |525     rol  ax,#4            ! bring lo-nibble to position
1a3:   24 0f           and    $0xf,%al          |526 nout: and  al,#0x0F       ! display one nibble
1a5:   27              daa                      |527     daa                   ! shorter conversion routine
1a6:   04 f0           add    $0xf0,%al         |528     add  al,#0xF0
1a8:   14 40           adc    $0x40,%al         |529     adc  al,#0x40         ! is now a hex char 0..9A..F
                                                |530 #endif
                                                |531 ; display - write byte in AL to console
                                                |532 ;       preserves all register contents
                                                |533 ;
```

— On sauvegarde le numéro de volume sur la pile (instruction d'adresse `0x9f`, soit ligne 263). On place les indicateurs du volume (qui font partie des paramètres, voir ligne 95) dans le registre `DH` (instruction d'adresse `0x0a0`, soit ligne 265). On fait pointer `SI` sur l'adresse LBA du dernier secteur à copier (instruction d'adresse `0x0a3`, soit ligne 266) et on appelle la routine `pread` de lecture à l'emplacement du pointeur (instruction `0xa6`, soit ligne 267).

La routine `pread` est définie à partir de l'instruction d'adresse `0x188`, soit ligne 506. On récupère un mot double (instruction d'adresse `0x188`, soit ligne 509). S'il est nul, on a terminé (instructions d'adresses `0x18a` et `0x18d`, soit lignes 510 et 511). Sinon on ajuste s'il s'agit d'un disque RAID (instruction d'adresse `0x18f`, soit ligne 512), ce qui n'est pas notre cas d'après la valeur du paramètre `raid` (ligne 88). On lit un secteur (instruction d'adresse `0x193`, soit ligne 513), grâce à la routine `disk_read` déjà étudiée ci-dessus. On se prépare pour le secteur suivant (instruction d'adresse `0x196`, soit ligne 515).

La constante `SECTOR_SIZE` est définie dans le fichier `lilo.h` :

```
181 #define SECTOR_SIZE 512 /* disk sector size */
```

— Si on a reçu un code d'erreur (instructions d'adresses `0xa9` à `0xb3`, soit lignes 269 à 271), on affiche un espace et on arrête l'ordinateur (instructions d'adresses `0xde` à `0xf3`, soit lignes 323 à 350).

— Sinon on restaure la valeur de `SI` (instruction d'adresse `0xb5`, soit ligne 273), on place la nouvelle valeur du décalage en mémoire dans `ES` (instructions d'adresses `0xb6` et `0xb9`, soit lignes 275 à 277) puis on copie le nombre de secteurs nécessaire à cet emplacement mémoire (instructions d'adresses `0xba` à `0xbf`, soit lignes 283 à 287).

Septième étape : Vérification de la signature de la seconde partie de LILO.- Une fois la seconde partie de `LILO` chargée en mémoire, on vérifie la signature de celle-ci :

```
                                        |289 ! Verify second stage loader signature
                                        |290
c1:   be 06 00        mov    $0x6,%si   |291     mov  si,#sig         ! pointer to signature area
c4:   89 f7           mov    %si,%di    |292     mov  di,si
c6:   b9 0a 00        mov    $0xa,%cx   |293     mov  cx,#length      ! number of bytes to compare
c9:   b4 9a           mov    $0x9a,%ah  |294     mov  ah,#0x9A        ! possible error code
                                        |295     repe
cb:   f3 a6           repz cmpsb %es:(%di),%ds:(%si) |296   cmpsb          ! check Signature 1 & 2
cd:   75 0f           jne    0xde       |297     jne  error           ! check Signature 2
                                        |298
                                        |299 #if SECOND_CHECK
                                        |300 /* it would be nice to re-incorporate this check */
cf:   b0 02           mov    $0x2,%al   |301     mov  al,#STAGE_SECOND ! do not touch AH (error code)
d1:   ae              scas   %es:(%di),%al |302     scasb
d2:   75 0a           jne    0xde       |303     jne  error
                                        |304 #endif
                                        |305
```

— On fait pointer `DI` sur la signature (instructions d'adtresses `0xc1` et `0xc4`, soit lignes 291 et 292), telle qu'elle apparaît dans les paramètres (ligne 82), on initialise `CX` avec le nombre de caractères à comparer (instruction d'adresse `0xc6`, soit ligne 293), la constante `length` étant définie ligne 86, on place le code d'erreur éventuel dans `AH` (instruction d'adresse `0xc9`, soit ligne 294) et on compare (instruction d'adresse `0xcb`, soit lignes 295 et 296). S'il n'y a pas concordance alors, comme pour les autres cas d'erreur, on arrête l'ordinateur (instruction d'instruction `0xcd`, soit ligne 297).

— On vérifie également que l'octet suivant est bien `0x02` (instructions d'adresses `0xcf` à `0xd2`, soit lignes 299 à 304).

La constante `STAGE_SECOND` est définie dans le fichier `lilo.h` :

```
246 #define STAGE_SECOND 2 /* second stage loader code */
```

Huitième étape : donner la main à la seconde partie de LILO.- Une fois la vérification correctement effectuée, on affiche un 'I' à l'écran et on donne la main à cette seconde partie de `LILO`, maintenant située en mémoire vive :

```
                                        |306 ! Start the second stage loader  DS=location of Params
                                        |307
d4:   06              push   %es         |308     push es              ! segment of second stage
d5:   55              push   %bp         |309     push bp              ! BP==0
                                        |310
d6:   b0 49           mov    $0x49,%al  |311     mov  al,#0x49        ! display an 'I'
d8:   e8 cf 00        call   0x1aa      |312     call display
                                        |313
db:   cb              lret               |314     retf                 ! Jump to ES:BP
```

ce qui se comprend facilement.

Début de la seconde partie de LILO.- Nous avons vu que la fin de la seconde partie de `LILO` se trouve sur le secteur 268 924 054 du second disque dur. Le début se trouve donc quelques secteurs avant. Par tâtonnement, on le trouve 32 secteurs avant, soit au secteur 268 924 035. Récupérons-en le premier secteur et désassemblons-le en comparant avec le fichier `second.S` des sources de `LILO` :

```
$ dd if=/dev/sdb skip=268924035 count=4 of=guenievre-lilo-2.dat od --format=x1 guenievre-lilo-2.dat
$ objdump -D -b binary -m i8086 -Maddr16,data16 guenievre-lilo-2.dat

Disassembly of section .data:


00000000 <.data>:
```

```
                                                    |1   #if 0
                                                    |2   /* second.S  -  LILO second stage boot
                                                    |       loader */
                                                    |3   Copyright 1992-1998 Werner Almesberger.
                                                    |4   Copyright 1999-2006 John Coffman.
                                                    |5   All rights reserved.
                                                    |6
                                                    |7   Licensed under the terms contained in
                                                    |    the file 'COPYING' in the
                                                    |8   source directory.
                                                    |9
                                                    |10  #endif
                                                    |
                                                    |116
                                                    |117         .text
                                                    |118
                                                    |119         .globl  _main
                                                    |120         .org    0
                                                    |121
   0:   eb 4e                   jmp    0x50        |122 _main: jmp     start
                                                    |123
                                                    |124 #if NO_FS || DEBUG_NEW
   2:   00 00                                       |125 firstseg:      dw        0
   4:   00 00                                       |
                                                    |134 #endif
                                                    |135
                                                    |136         .org    6
                                                    |137
                                                    |138 ! Boot device parameters. They are set
                                                    |      by the installer.
                                                    |139
                                                    |140 sig:           .ascii    "LILO"
                                                    |141 version:       .word     VERSION
                                                    |142 mapstamp:      .long     0
                                                    |143
                                                    |144 stage:         .word     STAGE_SECOND|
                                                    |        STAGE_SERIAL|STAGE_MENU|STAGE_BITMAP
                                                    |145
                                                    |146 port:  .byte   0         ; COM port
                                                    |               (0 = unused, 1 = COM1, etc.)
                                                    |147 sparam: .byte  0         ; serial port
                                                    |                 parameters (0 = unused)
                                                    |148
                                                    |149 timout: .word  0         ; input timeout
                                                    |150 delay:  .word  0         ; boot delay
                                                    |151 ms_len: .word  0         ; initial
                                                    |                    greeting message
                                                    |152
                                                    |153
                                                    |154 kt_cx:  .word  0         ; keyboard
                                                    |                    translation table
                                                    |155 kt_dx:  .word  0
                                                    |156 kt_al:  .byte  0
                                                    |157
                                                    |158 flag2:  .byte  0         ; second stage
                                                    |                             specific flags
                                                    |159
                                                    |160
                                                    |161 ! GDT for "high" loading
```

```
                                                |162
                                                |163         .align 16
                                                |164
                                                |165 gdt:    ; space for BIOS
                                                |166         .blkb  0x10
                                                |167         ; source
                                                |168         .word  0xffff     ; no limits
                                                |169         .byte  0
                                                |170         .word  LOADSEG>>4 ; start:
                                                |                              0x10000
                                                |171         .byte  0x93       ; permissions
                                                |172         .word  0          ; padding for
                                                |                              80286 mode :-(
                                                |173         ; destination
                                                |174         .word  0xffff     ; no limits
                                                |175         .word  0          ; start -
                                                |                         filled in by user
                                                |176         .byte  0
                                                |177         .byte  0x93       ; permissions
                                                |178         .word  0          ; padding for
                                                |                              80286 mode :-(
                                                |179         ; space for BIOS
                                                |180         .blkb  0x10
                                                |181
50:   fc                      cld               |182 start:  cld               ; only CLD in
                                                |                       the code; there is no STD
                                                |183 #if   ! NO_FS
51:   1e                      push   %ds        |184         push   ds
52:   0f a1                   pop    %fs        |185         pop    fs         ; address
                                                |                           parameters from here
                                                |186 #endif
                                                |
                                                |192         seg    cs
54:   2e 89 16 ea 24          mov    %dx,%cs:0x24ea |193      mov    [init_dx],dx ; save DX
                                                |                      passed in from first.S
                                                |194
59:   cd 12                   int    $0x12      |195         int    0x12       ; get memory
                                                |                              available
                                                |196         CHECK_FS
                                                |
5b:   c1 e0 06                shl    $0x6,%ax   |200         shl    ax,#6      ; convert to
                                                |                              paragraphs
5e:   2d 80 03                sub    $0x380,%ax |201         sub    ax,#Dataend/16
61:   8e c0                   mov    %ax,%es    |202         mov    es,ax      ; destination
                                                |                                address
63:   0e                      push   %cs        |203         push   cs
64:   1f                      pop    %ds        |204         pop    ds
65:   31 f6                   xor    %si,%si    |205         xor    si,si
67:   31 ff                   xor    %di,%di    |206         xor    di,di
69:   31 c0                   xor    %ax,%ax    |207         xor    ax,ax
6b:   b9 00 13                mov    $0x1300,%cx |208        mov    cx,#max_secondary/2
                                                |                  ; count of words to move
6e:   f3 a5                   rep movsw %ds:(%si),%es:(%di) |209  rep
                                                |210           movsw
70:   81 c7 00 0e             add    $0xe00,%di |211       add    di,#BSSstart-max_secondary
74:   b9 00 01                mov    $0x100,%cx |212         mov    cx,#BSSsize/2
77:   f3 ab                   rep    stos %ax,%es:(%di) |213 rep
                                                |214           stosw
79:   06                      push   %es        |215         push   es
7a:   68 7e 00                push   $0x7e      |216         push   #continue
7d:   cb                      lret              |217         retf               ; branch to
                                                |                          continue address
                                                |218 continue:
```

Nous n'allons pas analyser le (long) code de `LILO`. Tout ce qu'il suffit de savoir, pour l'instant, est qu'il ne quitte pas le mode réel.

14.2.4 Appel du noyau avec l'utilitaire de démarrage multiple

Revenons donc au noyau LINUX.

L'utilitaire de démarrage multiple ne donne pas la main à la première ligne de code de `vmlinuz` (que nous avons étudiée ci-dessus) mais au code commençant au deuxième « secteur ». Allons donc directement au début de ce deuxième secteur, c'est-à-dire à l'instruction d'adresse `0x200`, correspondant encore au fichier `arch/x86/boot/header.S` :

```
                                        |271
                                        |272     # offset 512, entry point
                                        |273
                                        |274     .globl  _start
                                        |275 _start:
                                        |276             # Explicitly enter this as bytes, or the assembler
                                        |277             # tries to generate a 3-byte jump here, which
                                        |                  causes
                                        |278             # everything else to push off to the wrong offset.
200:   eb 62                  jmp    0x264 |279             .byte   0xeb            # short (2-byte) jump
                                        |280             .byte   start_of_setup-1f
                                        |281 1:
                                        |282
```

La première instruction (d'adresse `0x200`, soit lignes 279 et 280) est un saut court, renvoyant à l'instruction d'adresse `0x264`, soit ligne 433. À cause de l'assembleur utilisé (lignes 276 à 278), on doit coder cette instruction directement en langage machine : sinon elle est traduite avec un code à trois octets et non de deux octets (par la partie optimiseur de l'assembleur).

Continuer à utiliser le fichier de désassemblage en commençant par le premier octet :

```
261:   30 1e 01 8c             xor    %bl,-0x73ff
265:   d8 8e c0 fc             fmuls  -0x340(%bp)
269:   8c d2                   mov    %ss,%dx
26b:   39 c2                   cmp    %ax,%dx
26d:   89 e2                   mov    %sp,%dx
```

ne nous donne pas grand chose : le désassembleur a, classiquement, perdu pied à cause des données ; en particulier, il ne trouve pas de début d'instruction en `0x264`. Utilisons donc l'option `-start-address=` de `objdump` :

```
$ objdump -D -b binary -mi8086 --start-address=0x264 ./sect1.com

./sect1.com:     file format binary


Disassembly of section .data:

00000264 <.data+0x264>:
 264:   8c d8                   mov    %ds,%ax
 266:   8e c0                   mov    %ax,%es
 268:   fc                      cld
 269:   8c d2                   mov    %ss,%dx
 26b:   39 c2                   cmp    %ax,%dx
 26d:   89 e2                   mov    %sp,%dx
 26f:   74 16                   je     0x287
 271:   ba 70 55                mov    $0x5570,%dx
 274:   f6 06 11 02 80          testb  $0x80,0x211
 279:   74 04                   je     0x27f
[...]
$
```

On trouve le code correspondant dans le fichier source :

```
                                        |431
                                        |432     .section ".entrytext", "ax"
                                        |433 start_of_setup:
                                        |434 # Force %es = %ds
264:   8c d8           mov    %ds,%ax   |435     movw %ds, %ax
266:   8e c0           mov    %ax,%es   |436     movw %ax, %es
268:   fc              cld              |437     cld
                                        |438
                                        |439 # Apparently some ancient versions of LILO invoked the
                                        |      kernel with %ss != %ds,
                                        |440 # which happened to work by accident for the old code.
                                        |      Recalculate the stack
                                        |441 # pointer if %ss is invalid.  Otherwise leave it alone,
                                        |      LOADLIN sets up the
                                        |442 # stack behind its own code, so we can't blindly put it
                                        |      directly past the heap.
                                        |443
269:   8c d2           mov    %ss,%dx   |444     movw    %ss, %dx
26b:   39 c2           cmp    %ax,%dx   |445     cmpw    %ax, %dx        # %ds == %ss?
26d:   89 e2           mov    %sp,%dx   |446     movw    %sp, %dx
26f:   74 16           je     0x287     |447     je      2f              # -> assume %sp is reasonably
                                        |                                     set
                                        |448
                                        |449     # Invalid %ss, make up a new stack
271:   ba b0 4e        mov    $0x4eb0,%dx |450     movw    $_end, %dx
274:   f6 06 11 02 80  testb  $0x80,0x211 |451     testb   $CAN_USE_HEAP, loadflags
279:   74 04           je     0x27f     |452     jz      1f
27b:   8b 16 24 02     mov    0x224,%dx |453     movw    heap_end_ptr, %dx
27f:   81 c2 00 02     add    $0x200,%dx |454 1:  addw    $STACK_SIZE, %dx
283:   73 02           jae    0x287     |455     jnc     2f
285:   31 d2           xor    %dx,%dx   |456     xorw    %dx, %dx        # Prevent wraparound
                                        |457
                                        |458 2:  # Now %dx should point to the end of our stack space
287:   83 e2 fc        and    $0xfffc,%dx |459     andw    $~3, %dx        # dword align (might as well...)
28a:   75 03           jne    0x28f     |460     jnz     3f
28c:   ba fc ff        mov    $0xfffc,%dx |461     movw    $0xfffc, %dx    # Make sure we're not zero
28f:   8e d0           mov    %ax,%ss   |462 3:  movw    %ax, %ss
291:   66 0f b7 e2     movzwl %dx,%esp  |463     movzwl  %dx, %esp       # Clear upper half of %esp
295:   fb              sti              |464     sti                     # Now we should have a working
                                        |                                  stack
                                        |465
                                        |466 # We will have entered with %cs = %ds+0x20, normalize %cs so
                                        |467 # it is on par with the other segments.
296:   1e              push   %ds       |468     pushw   %ds
297:   68 9b 02        push   $0x29b    |469     pushw   $6f
29a:   cb              lret             |470     lretw
                                        |471 6:
                                        |472
                                        |473 # Check signature at end of setup
29b:   66 81 3e ec 3b 55 aa   cmpl   $0x5a5aaa55,0x3bec |474     cmpl    $0x5a5aaa55, setup_sig
2a2:   5a 5a                            |
2a4:   75 17           jne    0x2bd     |475     jne     setup_bad
                                        |476
                                        |477 # Zero the bss
2a6:   bf f0 3b        mov    $0x3bf0,%di |478     movw    $__bss_start, %di
2a9:   b9 b3 4e        mov    $0x4eb3,%cx |479     movw    $_end+3, %cx
2ac:   66 31 c0        xor    %eax,%eax |480     xorl    %eax, %eax
2af:   29 f9           sub    %di,%cx   |481     subw    %di, %cx
2b1:   c1 e9 02        shr    $0x2,%cx  |482     shrw    $2, %cx
2b4:   f3 66 ab        rep stos %eax,%es:(%di) |483     rep; stosl
                                        |484
                                        |485 # Jump to C code (should not return)
2b7:   66 e8 5f 2a 00 00  calll  0x2d1c |486    calll    main
                                        |487
                                        |488 # Setup corrupt somehow...
                                        |489 setup_bad:
2bd:   66 b8 7f 03 00 00  mov    $0x37f,%eax |490     movl    $setup_corrupt, %eax
2c3:   66 e8 92 00 00 00       calll  0x35b |491     calll   puts
                                        |492     # Fall through...
                                        |493
                                        |494     .globl  die
                                        |495     .type   die, @function
```

```
                                                |496 die:
2c9:   f4              hlt                      |497     hlt
2ca:   e9 fc ff        jmp    0x2c9             |498     jmp     die
                                                |499
                                                |500     .size   die, .-die
                                                |501
                                                |502     .section ".initdata", "a"
                                                |503 setup_corrupt:
37f:   07                                       |504     .byte   7
380:   4e 6f 20 73 65 74 75 70                  |505     .string "No setup
388:   20 73 69 67 6e 61 74 75                  |                 signatu
390:   72 65 20 66 6f 75 6e 64                  |                re found
398:   2e 2e 2e 0a                             |                ...\n"
                                                |         .size   die, .-die
```

— On confond le segment supplémentaire (qui sera utilisé pour l'affichage des messages) avec le segment des données (instructions d'adresses `0x264` et `0x266`, soit lignes 435 et 436) et on indique que l'on progressera en avant lors des opérations sur les chaînes de caractères (instruction d'adresse `0x268`, soit ligne 437).

— On rectifie une erreur de certaines versions anciennes de `LILO` à propos du registre de pile (instructions d'adresses `0x269` à `0x285`, soit lignes 439 à 456).

N'insistons pas puisque ce n'est certainement pas ce qui se passe avec notre version.

— On initialise le pointeur de pile (instructions d'adresses `0x287` à `0x291`, soit lignes 458 à 463) et on permet les interruptions masquables (instruction d'adresse `0x295`, soit ligne 464), puisqu'on aura besoin d'utiliser les interruptions du BIOS.

— Comme nous l'avons déjà dit, et comme il est rappelé ligne 467, l'utilitaire de démarrage multiple nous a fait entrer avec `DS + 0x20` comme valeur du registre de segment `CS`.

On place comme valeur dans `CS` (instructions d'adresses `0x296` à `0x29a`, soit lignes 468 à 470) l'adresse de la ligne 471.

— Par convention des utilitaires de démarrage multiple, un secteur de démarrage doit se terminer par la signature `0x5a5aaa55`. S'il n'en est pas ainsi pour le nôtre (instructions d'adresses `0x29b` à `0x2a4`, soit lignes 473 à 475), on affiche un message d'erreur (instructions d'adresses `0x2bd` à `0x2c3`, soit lignes 488 à 491, et instructions d'adresses `0x37f` à `0x298`, soit lignes 503 à 505) et on arrête l'ordinateur (instruction d'adresse `0x2c9`, soit ligne 497).

— Si, par contre, la signature est bien présente, on initialise à zéro la section `bss` (*Block Started by Symbol*, instructions d'adresses `0x2a6` à `0x2b4`, soit lignes 477 à 483) de l'entête.

— On appelle alors la routine `main()` (instruction d'adresse `0x2b7`, soit ligne 486), que nous allons étudier maintenant.

14.3 Passage du mode réel au mode protégé

14.3.1 La routine `main()`

La routine `main()`, dont le code nous dit qu'elle commence à l'instruction d'adresse `0x2d1c`, est définie dans le fichier `arch/x86/boot/main.c` :

```
$ objdump -D -b binary -m i8086 -Maddr16,data16 --start-address=0x2d1c /boot/vmlinuz64-3.4.42

/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:

00002d1c <.data+0x2d1c>:
```

```
|1   /* -*- linux-c -*-
|2    *
|3    *   Copyright (C) 1991, 1992 Linus
|                    Torvalds
|4    *   Copyright 2007 rPath, Inc. -
|         All Rights Reserved
|5    *   Copyright 2009 Intel
|         Corporation; author H. Peter
|         Anvin
|6    *
|7    * This file is part of the Linux
|       kernel, and is made available
|       under
|8    *   the terms of the GNU General
|         Public License version 2.
|9    *
|10   * ---------------------------------
|---------------------------------- */
|11
|12  /*
|13   * Main module for the real-mode
|       kernel code
|14   */
|15
|16  #include "boot.h"

|134 void main(void)
|135 {
|136 /* First, copy the boot header into
|       the "zeropage" */
|137     copy_boot_params();
|
|22
|23 /*
|24  * Copy the header into the boot
|      parameter block.  Since this
|25  * screws up the old-style command
|      line protocol, adjust by
|26  * filling in the new-style command
|      line pointer instead.
|27  */
|28
|29 static void copy_boot_params(void)
|30 {
|31       struct old_cmdline {
|32               u16 cl_magic;
|33               u16 cl_offset;
|34       };
|35       const struct old_cmdline * const
|         oldcmd =
|36               (const struct old_cmdline
|                *)OLD_CL_ADDRESS;
|37
|38       BUILD_BUG_ON(sizeof boot_params
|                    != 4096);
 2d1c:   66 55                   push   %ebp
|39       memcpy(&boot_params.hdr, &hdr,
|                sizeof hdr);
```

```
2d1e:   66 89 e5                mov    %esp,%ebp                    |
2d21:   66 57                   push   %edi                         |
2d23:   66 56                   push   %esi                         |
2d25:   66 83 e4 f0             and    $0xfffffff0,%esp             |
2d29:   66 83 ec 60             sub    $0x60,%esp                   |
2d2d:   66 be f1 01 00 00       mov    $0x1f1,%esi                  |
2d33:   66 bf 81 40 00 00       mov    $0x4081,%edi                 |
2d39:   66 b9 73 00 00 00       mov    $0x73,%ecx                   |
2d3f:   f3 a4                   rep movsb %ds:(%si),%es:(%di)       |
                                                                    |40
2d41:   66 83 3e b8 40 00       cmpl   $0x0,0x40b8                  |41      if (!boot_params.hdr.cmd_line_ptr
                                                                    |                &&
                                                                    |42            oldcmd->cl_magic ==
                                                                    |                              OLD_CL_MAGIC)
                                                                    |                 {
                                                                    |43                /* Old-style command
                                                                    |                       line protocol. */
                                                                    |44                u16 cmdline_seg;
                                                                    |45
                                                                    |46                /* Figure out if the
                                                                    |          command line falls in the region
                                                                    |47                      of memory that an
                                                                    |          old kernel would have copied up
                                                                    |48                      to 0x90000... */
2d47:   75 2b                   jne    0x2d74                       |
2d49:   81 3e 20 00 3f a3       cmpw   $0xa33f,0x20                 |49                if (oldcmd->cl_offset <
                                                                    |         boot_params.hdr.setup_move_size)
2d4f:   75 23                   jne    0x2d74                       |
2d51:   66 0f b7 16 22 00       movzwl 0x22,%edx                    |50                      cmdline_seg =
                                                                    |                                  ds();
                                                                    |51                else
2d57:   66 b8 00 90 ff ff       mov    $0xffff9000,%eax             |52                      cmdline_seg =
                                                                    |                                0x9000;
2d5d:   3b 16 a2 40             cmp    0x40a2,%dx                   |
2d61:   73 02                   jae    0x2d65                       |
2d63:   8c d8                   mov    %ds,%ax                      |53         boot_params.hdr.cmd_line_ptr =
2d65:   66 0f b7 c0             movzwl %ax,%eax                     |
2d69:   66 c1 e0 04             shl    $0x4,%eax                    |54                    (cmdline_seg << 4)
                                                                    |            + oldcmd->cl_offset;
2d6d:   66 01 d0                add    %edx,%eax                    |
2d70:   66 a3 b8 40             mov    %eax,0x40b8                  |
                                                                    |55       }
           }
                                                                    |56 }
                                                                    |57
                                                                    |
                                                                    |138
                                                                    |139     /* Initialize the early-boot
                                                                    |            console */
2d74:   66 e8 b3 df ff ff       calll  0xd2d                        |140     console_init();
2d7a:   66 ba 44 30 00 00       mov    $0x3044,%edx                 |
2d80:   66 a1 b8 40             mov    0x40b8,%eax                  |
2d84:   66 e8 f6 d8 ff ff       calll  0x680                        | cmdline_find_option_bool()
2d8a:   66 85 c0                test   %eax,%eax                    |
2d8d:   74 0c                   je     0x2d9b                       |141     if (cmdline_find_option_bool(
                                                                    |              "debug"))
2d8f:   66 b8 4a 30 00 00       mov    $0x304a,%eax                 | /* addr of "early console..." */
2d95:   66 e8 c0 d5 ff ff       calll  0x35b                        |142             puts("early console in
                                                                    |                    setup code\n");
                                                                    |143
                                                                    |144     /* End of heap check */
                                                                    |145     init_heap();
2d9b:   80 3e a1 40 00          cmpb   $0x0,0x40a1                  | if (...
2da0:   79 28                   jns    0x2dca                       |    "no"
2da2:   67 66 8d 84 24 00 fe    lea    -0x200(%eax,%eax,1),%eax     | asm("leal ...
2da9:   ff ff                                                       |
2dab:   66 0f b7 16 b4 40       movzwl 0x40b4,%edx                  |
2db1:   66 81 c2 00 02 00 00    add    $0x200,%edx                  |
2db8:   66 39 c2                cmp    %eax,%edx                    |
2dbb:   77 07                   ja     0x2dc4                       |
2dbd:   66 89 16 98 3b          mov    %edx,0x3b98                  |
2dc2:   eb 12                   jmp    0x2dd6                       |
2dc4:   66 a3 98 3b             mov    %eax,0x3b98                  |
2dc8:   eb 0c                   jmp    0x2dd6                       |
```

```
2dca:   66 b8 67 30 00 00       mov    $0x3067,%eax                 | /* addr of "WARNING: Ancient */
                                                                    | /* bootloader..." */
2dd0:   66 e8 85 d5 ff ff       calll  0x35b                        | /* puts it */
                                                                    |146
                                                                    |147     /* Make sure we have all the
                                                                    |            proper CPU support */
2dd6:   66 e8 d0 d9 ff ff       calll  0x7ac                        | validate_cpu()
2ddc:   66 85 c0                test   %eax,%eax                    |148     if (validate_cpu()) {
2ddf:   74 12                   je     0x2df3                       |        "yes"
2de1:   66 b8 a8 30 00 00       mov    $0x30a8,%eax                 | /* addr of "Unable to boot..." */
2de7:   66 e8 6e d5 ff ff       calll  0x35b                        |149             puts("Unable to boot -
                                                                    |     please use a kernel appropriate "
                                                                    |150                  "for your CPU.\n");
2ded:   66 e8 d6 d4 ff ff       calll  0x2c9                        |151             die();
                                                                    |152     }
                                                                    |153
                                                                    |154     /* Tell the BIOS what CPU mode
                                                                    |            we intend to run in. */
                                                                    |155     set_bios_mode();
2df3:   67 66 8d 44 24 34       lea    0x34(%eax,%eax,1),%eax       |
2df9:   66 e8 11 eb ff ff       calll  0x1910                       | initregs(&ireg);
2dff:   67 c7 44 24 50 00 ec    movw   $0xec00,0x50(%eax,%eax,1)    |
2e06:   67 c7 44 24 44 02 00    movw   $0x2,0x44(%eax,%eax,1)       |
2e0d:   66 31 c9                xor    %ecx,%ecx                    |
2e10:   67 66 8d 54 24 34       lea    0x34(%eax,%eax,1),%edx       |
2e16:   66 b8 15 00 00 00       mov    $0x15,%eax                   |
2e1c:   66 e8 e6 d6 ff ff       calll  0x508                        | intcall(0x15, &ireg, NULL);
                                                                    |156
                                                                    |157     /* Detect memory layout */
2e22:   66 e8 00 e2 ff ff       calll  0x1028                       |158     detect_memory();
                                                                    |159
                                                                    |160     /* Set keyboard repeat rate
                                                                    |      (why?) and query the lock flags */
                                                                    |161     keyboard_init();
                                                                    |keyboard_set_repeat:
2e28:   67 66 8d 44 24 34       lea    0x34(%eax,%eax,1),%eax       |
2e2e:   66 e8 dc ea ff ff       calll  0x1910                       | initregs(&ireg);
2e34:   67 c7 44 24 50 05 03    movw   $0x305,0x50(%eax,%eax,1)     |
2e3b:   66 31 c9                xor    %ecx,%ecx                    |
2e3e:   67 66 8d 54 24 34       lea    0x34(%eax,%eax,1),%edx       |
2e44:   66 b8 16 00 00 00       mov    $0x16,%eax                   |
2e4a:   66 e8 b8 d6 ff ff       calll  0x508                        | intcall(...)
                                                                    |162
                                                                    |163     /* Query MCA information */
2e50:   66 e8 62 e1 ff ff       calll  0xfb8                        |164     query_mca();
                                                                    |165
                                                                    |166     /* Query Intel SpeedStep (IST)
                                                                    |            information */
                                                                    |167     query_ist();
2e56:   66 83 3e 60 3e 05       cmpl   $0x5,0x3e60                  |
2e5c:   7e 5d                   jle    0x2ebb                       |
2e5e:   67 66 8d 44 24 08       lea    0x8(%eax,%eax,1),%eax        |
2e64:   66 e8 a6 ea ff ff       calll  0x1910                       | initregs(&ireg);
2e6a:   67 c7 44 24 24 80 e9    movw   $0xe980,0x24(%eax,%eax,1)    |
2e71:   67 66 c7 44 24 1c 43    movl   $0x47534943,0x1c(%eax,%eax,1) |
2e78:   49 53 47                                                    |
2e7b:   67 66 8d 4c 24 34       lea    0x34(%eax,%eax,1),%ecx       |
2e81:   67 66 8d 54 24 08       lea    0x8(%eax,%eax,1),%edx        |
2e87:   66 b8 15 00 00 00       mov    $0x15,%eax                   |
2e8d:   66 e8 75 d6 ff ff       calll  0x508                        | intcall(...)
2e93:   67 66 8b 44 24 50       mov    0x50(%eax,%eax,1),%eax       |
2e99:   66 a3 f0 3e             mov    %eax,0x3ef0                  |
2e9d:   67 66 8b 44 24 44       mov    0x44(%eax,%eax,1),%eax       |
2ea3:   66 a3 f4 3e             mov    %eax,0x3ef4                  |
2ea7:   67 66 8b 44 24 4c       mov    0x4c(%eax,%eax,1),%eax       |
2ead:   66 a3 f8 3e             mov    %eax,0x3ef8                  |
2eb1:   67 66 8b 44 24 48       mov    0x48(%eax,%eax,1),%eax       |
2eb7:   66 a3 fc 3e             mov    %eax,0x3efc                  |
                                                                    |168
                                                                    |169     /* Query APM information */
                                                                    |170 #if defined(CONFIG_APM) ||
                                                                    |        defined(CONFIG_APM_MODULE)
                                                                    |171     query_apm_bios();
                                                                    |172 #endif
```

```
                                                        |173
                                                        |174     /* Query EDD information */
                                                        |175 #if defined(CONFIG_EDD) ||
                                                        |        defined(CONFIG_EDD_MODULE)
                                                        |176     query_edd();
                                                        |177 #endif
                                                        |178
                                                        |179     /* Set the video mode */
2ebb:   66 e8 94 ee ff ff       calll  0x1d55           |180     set_video();
                                                        |181
                                                        |182     /* Do the last things and
                                                        |           invoke protected mode */
2ec1:   66 e8 e9 e2 ff ff       calll  0x11b0           |183    go_to_protected_mode();
                                                        |184 }
```

Nous n'allons pas entrer dans les détails. Il s'agit d'initialiser un certain nombre de choses :

— Les 115 (= `0x73`) octets des paramètres de démarrage sont recopiés (instructions d'adresses `0x2d1c` à `0x2d70`, soit lignes 136 et 137) à partir du décalage `0x1f1` du fichier de code vers la ligne d'adresse `0x4081` en mémoire vive.

— La console est initialisée (instruction d'adresse `0x2d74`, soit ligne 140), avec un message si on se trouve dans le mode `debug` (instructions d'adresses `0x2d7a` à `0x2d95`, soit lignes 139 à 142).

— La pile est initialisée (instructions d'adresses `0x2d9b` à `0x2dd0`, soit lignes 144 et 145).

— On vérifie que les options sont valides pour le microprocesseur de l'ordinateur sur lequel on se trouve (instructions d'adresses `0x2dd6` à `0x2ded`, soit lignes 147 à 152).

— On spécifie au BIOS dans quel mode on veut se trouver (instructions d'adresses `0x2df3` à `0x2e1c`, soit lignes 154 à 156), ce qui a une incidence si on veut utiliser l'architecture x86-64.

La fonction `set_bios_mode()` est définie dans le même fichier (lignes 100 à 113).

— On détecte la disposition de la mémoire (instruction d'adresse `0x2e22`, soit lignes 157 et 158).

— On initialise la vitesse de répétition des touches du clavier (instructions d'adresses `0x2e28` à `0x2e4a`, soit lignes 160 à 162).

— On s'enquiert des informations MCA (*Micro Channel Architecture*, instruction d'adresse `0x2e50`, soit lignes 163 et 164).

— On s'enquiert des informations IST (instructions d'adresses`0x2e56` à `0x2eb7`, soit lignes 166 à 168).

— On s'enquiert éventuellement des informations APM (lignes 169 à 172), ce qui n'est pas le cas ici.

— On s'enquiert éventuellement des informations EDD (*Enhanced Disk Drive*, lignes 174 à 177), ce qui n'est pas le cas ici.

— On se place dans un mode d'affichage (instruction d'adresse `0x2ebb`, soit lignes 179 et 180).

— On appelle la routine de passage du mode réel au mode protégé (instruction d'adresse `0x2ec1`, soit lignes 182 et 183).

14.3.2 Paramètres de démarrage

Les 115 octets de paramètres de démarrage, recopiés depuis le décalage `0x1f1` du fichier de code à l'emplacement mémoire `0x4081`, sont définis dans le fichier `arch/x86/boot/header.S` :

```
                |28
                |29 BOOTSEG            = 0x07C0               /* original address of boot-sector */
                |30 SYSSEG             = 0x1000               /* historical load address >> 4 */
                |31
                |32 #ifndef SVGA_MODE
                |33 #define SVGA_MODE ASK_VGA
                |34 #endif
                |35
                |36 #ifndef ROOT_RDONLY
                |37 #define ROOT_RDONLY 1
                |38 #endif
                |39
                |
                |98
                |99  #ifdef CONFIG_EFI_STUB
                |
                |253 #endif /* CONFIG_EFI_STUB */
                |254
                |255     # Kernel attributes; used by setup.  This is part 1 of the
                |256     # header, from the old boot sector.
                |257
                |258     .section ".header", "a"
                |259     .globl  sentinel
                |260 sentinel:   .byte 0xff, 0xff        /* Used to detect broken loaders */
                |261
                |262     .globl  hdr
                |263 hdr:
1f1: 1d         |264 setup_sects: .byte 0                /* Filled in by build.c */
1f2: 01 00      |265 root_flags: .word ROOT_RDONLY
1f4: fe 02 05 00|266 syssize:    .long 0                 /* Filled in by build.c */
1f8: 00 00      |267 ram_size:   .word 0                 /* Obsolete */
1fa: ff ff      |268 vid_mode:   .word SVGA_MODE
1fc: 00 00      |269 root_dev:   .word 0                 /* Filled in by build.c */
1fe: 55 aa      |270 boot_flag:  .word 0xAA55
                |
                |283     # Part 2 of the header, from the old setup.S
                |284
202: 48 64 72 53|285             .ascii  "HdrS"          # header signature
206: 0a 02      |286             .word   0x020c          # header  version number (>= 0x0105)
                |287                                     # or else old loadlin-1.5 will fail)
                |288             .globl realmode_swtch
208: 00 00 00 00|289 realmode_swtch:     .word   0, 0    # default_switch, SETUPSEG
20c: 00 10      |290 start_sys_seg:      .word   SYSSEG  # obsolete and meaningless, but just
                |291                                     # in case something decided to "use" it
20e: 50 30      |292             .word   kernel_version-512 # pointing to kernel version string
                |293                                     # above section of header is compatible
                |294                                     # with loadlin-1.5 (header v1.5). Don't
                |295                                     # change it.
                |296
210: 00         |297 type_of_loader:     .byte   0       # 0 means ancient bootloader, newer
                |298                                     # bootloaders know to change this.
                |399                                     # See Documentation/x86/boot.txt for
                |300                                     # assigned ids
                |301
                |302 # flags, unused bits must be zero (RFU) bit within loadflags
                |303 loadflags:
211: 01         |304             .byte   LOADED_HIGH     # The kernel is to be loaded high
                |305
212: 00 80      |306 setup_move_size:    .word  0x8000   # size to move, when setup is not
                |307                                     # loaded at 0x90000. We will move setup
                |308                                     # to 0x90000 then just before jumping
                |309                                     # into the kernel. However, only the
                |310                                     # loader knows how much data behind
                |311                                     # us also needs to be loaded.
                |312
                |313 code32_start:                       # here loaders can put a different
                |314                                     # start address for 32-bit code.
214: 00 00 10 00|315             .long     0x100000      # 0x100000 = default for big kernel
                |316
```

```
218: 00 00 00 00 |317 ramdisk_image: .long  0            # address of loaded ramdisk image
                 |318                                   # Here the loader puts the 32-bit
                 |319                                   # address where it loaded the image.
                 |320                                   # This only will be read by the kernel.
                 |321
21c: 00 00 00 00 |322 ramdisk_size:  .long  0            # its size in bytes
                 |323
                 |324 bootsect_kludge:
220: 00 00 00 00 |325             .long   0             # obsolete
                 |326
224: b0 4e       |327 heap_end_ptr: .word _end+STACK_SIZE-512
                 |328                                   # (Header version 0x0201 or later)
                 |329                                   # space from here (exclusive) down to
                 |330                                   # end of setup code can be used by setup
                 |331                                   # for local heap purposes.
                 |332
                 |333 ext_loader_ver:
226: 00          |334             .byte   0             # Extended boot loader version
                 |335 ext_loader_type:
227: 00          |336             .byte   0             # Extended boot loader type
                 |337
228: 00 00 00 00 |338 cmd_line_ptr: .long 0              # (Header version 0x0202 or later)
                 |339                                   # If nonzero, a 32-bit pointer
                 |340                                   # to the kernel command line.
                 |341                                   # The command line should be
                 |342                                   # located between the start of
                 |343                                   # setup and the end of low
                 |344                                   # memory (0xa0000), or it may
                 |345                                   # get overwritten before it
                 |346                                   # gets read.  If this field is
                 |347                                   # used, there is no longer
                 |348                                   # anything magical about the
                 |349                                   # 0x90000 segment; the setup
                 |350                                   # can be located anywhere in
                 |351                                   # low memory 0x10000 or higher.
                 |352
22c: ff ff ff 7f |353 ramdisk_max:  .long 0x7fffffff
                 |354                                   # (Header version 0x0203 or later)
                 |355                                   # The highest safe address for
                 |356                                   # the contents of an initrd
                 |357                                   # The current kernel allows up to 4 GB,
                 |358                                   # but leave it at 2 GB to avoid
                 |359                                   # possible bootloader bugs.
                 |360
230: 00 00 00 01 |361 kernel_alignment: .long CONFIG_PHYSICAL_ALIGN   #physical addr alignment
                 |362                                                 #required for protected mode
                 |363                                                 #kernel
                 |364 #ifdef CONFIG_RELOCATABLE
                 |365 relocatable_kernel:   .byte 1
                 |366 #else
234: 00          |367 relocatable_kernel:   .byte 0
                 |368 #endif
235: 15          |369 min_alignment:        .byte MIN_KERNEL_ALIGN_LG2 # minimum alignment
                 |370
                 |371 xloadflags:
                 |372 #ifdef CONFIG_X86_64
                 |373 # define XLF0 XLF_KERNEL_64                 /* 64-bit kernel */
                 |374 #else
                 |375 # define XLF0 0
                 |376 #endif
                 |377
                 |378 #if defined(CONFIG_RELOCATABLE) && defined(CONFIG_X86_64)
                 |379   /* kernel/boot_param/ramdisk could be loaded above 4g */
                 |380 # define XLF1 XLF_CAN_BE_LOADED_ABOVE_4G
                 |381 #else
                 |382 # define XLF1 0
                 |383 #endif
                 |384
                 |385 #ifdef CONFIG_EFI_STUB
                 |386 # ifdef CONFIG_X86_64
                 |387 #  define XLF23 XLF_EFI_HANDOVER_64         /* 64-bit EFI handover ok */
                 |388 # else
                 |389 #  define XLF23 XLF_EFI_HANDOVER_32         /* 32-bit EFI handover ok */
                 |390 # endif
```

```
                |391 #else
                |392 # define XLF23 0
                |393 #endif
236: 00 00      |394                     .word XLF0 | XLF1 | XLF23
                |395
238: ff 07 00 00 |396 cmdline_size: .long COMMAND_LINE_SIZE-1     #length of the command line,
                |397                                             #added with boot protocol
                |398                                             #version 2.06
                |499
23c: 00 00 00 00 |400 hardware_subarch:   .long 0                 # subarchitecture, added with 2.07
                |401                                             # default to 0 for normal x86 PC
                |402
240: 00 00 00 00 |403 hardware_subarch_data: .quad 0
     00 00 00 00 |
                |404
248: 58 02 00 00 |405 payload_offset:     .long ZO_input_data
24c: 98 df 4f 00 |406 payload_length:     .long ZO_z_input_len
                |407
250: 00 00 00 00 |408 setup_data:         .quad 0                 # 64-bit physical pointer to
     00 00 00 00 |
                |409                                             # single linked list of
                |410                                             # struct setup_data
                |411
258: 00 00 00 01 |412 pref_address:       .quad LOAD_PHYSICAL_ADDR      # preferred load addr
     00 00 00 00 |
                |413
                |414 #define ZO_INIT_SIZE (ZO__end - ZO_startup_32 + ZO_z_extract_offset)
                |415 #define VO_INIT_SIZE (VO__end - VO__text)
                |416 #if ZO_INIT_SIZE > VO_INIT_SIZE
                |417 #define INIT_SIZE ZO_INIT_SIZE
                |418#else
                |419 #define INIT_SIZE VO_INIT_SIZE
                |420 #endif
260: 00 30      |421 init_size:          .long INIT_SIZE         # kernel initialization size
                |422 handover_offset:
                |423 #ifdef CONFIG_EFI_STUB
                |424                     .long 0x30              # offset to the handover
                |425                                             # protocol entry point
                |426 #else
262: eb 00      |427                     .long 0
                |428 #endif
                |429
                |430 # End of setup header
```

14.3.3 Routine de passage au mode protégé

Ne détaillons que la dernière action, le passage au mode protégé. La dernière fonction appelée, à savoir `go_to_protected_mode()`, commençant à l'adresse `0x11b0`, est définie dans le fichier `arch/x86/boot/pm.c` :

```
objdump -D -b binary -mi386 -Maddr16,data16 --start-address=0x11b0 /boot/vmlinuz64-3.4.42

/boot/vmlinuz64-3.4.42:     file format binary

Disassembly of section .data:

000011b0 <.data+0x11b0>:
                                                |10
                                                |11  /*
                                                |12   * Prepare the machine for transition to
                                                |       protected mode.
                                                |13   */
                                                |14
                                                |
                                                |100
                                                |101 /*
                                                |102  * Actual invocation sequence
                                                |103  */
    11b0:  66 53                push   %ebx     |104 void go_to_protected_mode(void)
    11b2:  66 83 ec 08          sub    $0x8,%esp|
                                                |105 {
                                                |106     /* Hook before leaving real mode, also
                                                |           disables interrupts */
                                                |107     realmode_switch_hook();
                                                |
                                                |17
                                                |18  /*
                                                |19   * Invoke the realmode switch hook if present;
                                                |       otherwise
                                                |20   * disable all interrupts.
                                                |21   */
                                                |22  static void realmode_switch_hook(void)
                                                |23  {
    11b6:  66 83 3e 98 40 00    cmpl   $0x0,0x4098|24      if (boot_params.hdr.realmode_swtch) {
    11bc:  74 06                je     0x11c4   |
    11be:  ff 1e 98 40          lcall  *0x4098  |25              asm volatile("lcallw *%0"
                                                |26                           : : "m"
                                                |                 (boot_params.hdr.realmode_swtch)
                                                |27                           : "eax", "ebx", "ecx",
                                                |                             "edx");
    11c2:  eb 07                jmp    0x11cb   |
                                                |28      } else {
    11c4:  fa                   cli             |29              asm volatile("cli");
    11c5:  b0 80                mov    $0x80,%al|30              outb(0x80, 0x70); /* Disable NMI */
    11c7:  e6 70                out    %al,$0x70|
    11c9:  e6 80                out    %al,$0x80|31              io_delay(); /* defined in "boot.h" */
                                                |32      }
                                                |33  }
                                                |34
                                                |
                                                |108
                                                |109     /* Enable the A20 gate */
    11cb:  66 e8 49 f2 ff ff    calll  0x41a    |110     if (enable_a20()) {
    11d1:  66 85 c0             test   %eax,%eax|
    11d4:  74 12                je     0x11e8   |     'no'
                                                |
    11d6:  66 b8 e8 30 00 00    mov    $0x30e8,%eax|111             puts("A20 gate not responding, unable
                                                |                       to boot...\n");
    11dc:  66 e8 79 f1 ff ff    calll  0x35b    |
    11e2:  66 e8 e1 f0 ff ff    calll  0x2c9    |112             die();
                                                |113     }
                                                |114
                                                |115     reset_coprocessor();
                                                |
                                                |45
                                                |46  /*
                                                |47   * Reset IGNNE# if asserted in the FPU.
```

```
                                                |48   */
                                                |49  static void reset_coprocessor(void)
                                                |50  {
11e8:  66 31 c0              xor    %eax,%eax   |51      outb(0, 0xf0);
11eb:  e6 f0                 out    %al,$0xf0   |
11ed:  e6 80                 out    %al,$0x80   |52      io_delay();
11ef:  e6 f1                 out    %al,$0xf1   |53      outb(0, 0xf1);
11f1:  e6 80                 out    %al,$0x80   |54      io_delay();
                                                |55  }
                                                |
                                                |116
                                                |117     /* Mask all interrupts in the PIC */
                                                |118     mask_all_interrupts();
                                                |
                                                |35  /*
                                                |36   * Disable all interrupts at the legacy PIC.
                                                |37   */
                                                |38  static void mask_all_interrupts(void)
                                                |39  {
11f3:  b0 ff                 mov    $0xff,%al   |40      outb(0xff, 0xa1);   /* Mask all interrupts
                                                |                              on the secondary PIC */
11f5:  e6 a1                 out    %al,$0xa1   |
11f7:  e6 80                 out    %al,$0x80   |41      io_delay();
11f9:  b0 fb                 mov    $0xfb,%al   |42      outb(0xfb, 0x21);   /* Mask all but cascade
                                                |                              on the primary PIC */
11fb:  e6 21                 out    %al,$0x21   |
11fd:  e6 80                 out    %al,$0x80   |43      io_delay();
                                                |44  }
                                                |
                                                |119
                                                |120     /* Actual transition to protected mode... */
                                                |121     setup_idt();
                                                |
                                                |91
                                                |92  /*
                                                |93   * Set up the IDT
                                                |94   */
                                                |95  static void setup_idt(void)
                                                |96  {
                                                |97      static const struct gdt_ptr null_idt = {0, 0};

11ff:  66 0f 01 1e 20 31     lidtl  0x3120       |98      asm volatile("lidtl %0" : : "m" (null_idt));
                                                |99  }
                                                |
                                                |122     setup_gdt();
                                                |
                                                |56
                                                |57  /*
                                                |58   * Set up the GDT
                                                |59   */
                                                |60
                                                |61  struct gdt_ptr {
                                                |62      u16 len;
                                                |63      u32 ptr;
                                                |64  } __attribute__((packed));
                                                |65
                                                |66  static void setup_gdt(void)
                                                |67  {
                                                |68      /* There are machines which are known to not
                                                |           boot with the GDT
                                                |69         being 8-byte unaligned.  Intel recommends
                                                |           16 byte alignment. */
                                                |70     static const u64 boot_gdt[]
                                                |        __attribute__((aligned(16))) = {
                                                |71              /* CS: code, read/execute, 4 GB,
                                                |        base 0 */
                                                |72              [GDT_ENTRY_BOOT_CS] =
                                                |        GDT_ENTRY(0xc09b, 0, 0xfffff),
                                                |73              /* DS: data, read/write, 4 GB,
                                                |        base 0 */
                                                |74              [GDT_ENTRY_BOOT_DS] =
                                                |        GDT_ENTRY(0xc093, 0, 0xfffff),
                                                |75              /* TSS: 32-bit tss, 104 bytes,
```

```
                                                    |                  base 4096 */
                                                    |76             /* We only have a TSS here to
                                                    |                  keep Intel VT happy;
                                                    |77                we don't actually use it for
                                                    |                  anything. */
                                                    |78             [GDT_ENTRY_BOOT_TSS] =
                                                    |         GDT_ENTRY(0x0089, 4096, 103),
                                                    |79       };
                                                    |80       /* Xen HVM incorrectly stores a pointer
                                                    |            to the gdt_ptr, instead
                                                    |81          of the gdt_ptr contents.  Thus, make
                                                    |            it static so it will
                                                    |82          stay in memory, at least long enough
                                                    |            that we switch to the
                                                    |83          proper kernel GDT. */
                                                    |84      static struct gdt_ptr gdt;
                                                    |85

1205:  c7 06 40 3c 27 00       movw   $0x27,0x3c40  |86      gdt.len = sizeof(boot_gdt)-1;
120b:  8c da                   mov    %ds,%dx       |87      gdt.ptr = (u32)&boot_gdt + (ds() << 4);
120d:  66 0f b7 d2             movzwl %dx,%edx      |
1211:  66 c1 e2 04             shl    $0x4,%edx     |
1215:  67 66 8d 82 30 31 00    lea    0x3130(%edx),%eax |
                                                    |88
121c:  00                                           |
121d:  66 a3 42 3c             mov    %eax,0x3c42   |
1221:  66 0f 01 16 40 3c       lgdtl  0x3c40        |89      asm volatile("lgdtl %0" : : "m" (gdt));
                                                    |90  }
                                                    |
1227:  66 81 c2 90 3e 00 00    add    $0x3e90,%edx  |123     protected_mode_jump(
                                                    |        boot_params.hdr.code32_start,
                                                    |124 (u32)&boot_params + (ds() << 4));
122e:  66 a1 a4 40             mov    0x40a4,%eax   |
1232:  66 e8 00 00 00 00       calll  0x1238        |
                                                    |125 }
```

— Les interruptions sont désactivées, y compris les interruptions matérielles non masquables (instructions d'adresses `0x11b0` à `0x11c9`, soit lignes 106 et 107, renvoyant à la fonction définie lignes 18 à 33).

— La broche `A20` du microprocesseur est activée (instruction d'adresse `0x11cb`, soit ligne 110). Si on n'y parvient pas, on affiche un message (instructions d'adresses `0x11d1` à `0x11dc`, soit ligne 111) et on propose un redémarrage (instruction d'adresse `0x11e2`, soit ligne 112).

Le code de `enable_a20()` se trouve dans le fichier `a20.o` qui, lui-même, commence par l'appel de `empty_8042()`, que l'on retrouve dans l'image du noyau :

```
$ objdump -d -mi386 -Maddr16,data16 /usr/src/linux/arch/x86/boot/a20.o|less

/usr/src/linux/arch/x86/boot/a20.o:     file format elf32-i386

Disassembly of section .text:

00000000 <empty_8042>:
   0:   66 ba a1 86 01 00       mov    $0x186a1,%edx
   6:   66 b9 20 00 00 00       mov    $0x20,%ecx
   c:   eb 1a                   jmp    28 <empty_8042+0x28>
   e:   e6 80                   out    %al,$0x80
  10:   e4 64                   in     $0x64,%al
  12:   3c ff                   cmp    $0xff,%al
  14:   75 04                   jne    1a <empty_8042+0x1a>
  16:   66 49                   dec    %ecx
  18:   74 12                   je     2c <empty_8042+0x2c>
  1a:   a8 01                   test   $0x1,%al
```

```
  1c:   74 06                   je     24 <empty_8042+0x24>
  1e:   e6 80                   out    %al,$0x80
  20:   e4 60                   in     $0x60,%al
  22:   eb 04                   jmp    28 <empty_8042+0x28>
  24:   a8 02                   test   $0x2,%al
  26:   74 0a                   je     32 <empty_8042+0x32>
  28:   66 4a                   dec    %edx
  2a:   75 e2                   jne    e <empty_8042+0xe>
  2c:   66 83 c8 ff             or     $0xffffffff,%eax
  30:   66 c3                   retl
  32:   66 31 c0                xor    %eax,%eax
  35:   66 c3                   retl
```

On voit facilement qu'il s'agit du code obtenu en compilant le code C :

```
static int empty_8042(void)
{
        u8 status;
        int loops = MAX_8042_LOOPS;
        int ffs   = MAX_8042_FF;

        while (loops--) {
                io_delay();

                status = inb(0x64);
                if (status == 0xff) {
                        /* FF is a plausible, but very unlikely status */
                        if (!--ffs)
                                return -1; /* Assume no KBC present */
                }
                if (status & 1) {
                        /* Read and discard input data */
                        io_delay();
                        (void)inb(0x60);
                } else if (!(status & 2)) {
                        /* Buffers empty, finished! */
                        return 0;
                }
        }

        return -1;
}
```

On en trouve l'emplacement de la façon suivante :

```
$ hexdump -C  /boot/vmlinuz64-3.4.42 |
  grep '66[[:space:]]*ba[[:space:]]*a1[[:space:]]*86[[:space:]]*01[[:space:]]*00'

000003a0  66 ba a1 86 01 00 66 b9  20 00 00 00 eb 1a e6 80  |f.....f. .......|

$ objdump -D -b binary -m i8086 -Maddr16,data16 --start-address=0x3a0
                                                    /boot/vmlinuz64-3.4.42|less
/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:
                                                     empty_8042
```

```
000003a0 <.data+0x3a0>:
     3a0:       66 ba a1 86 01 00       mov    $0x186a1,%edx
     3a6:       66 b9 20 00 00 00       mov    $0x20,%ecx
     3ac:       eb 1a                   jmp    0x3c8
     3ae:       e6 80                   out    %al,$0x80
     3b0:       e4 64                   in     $0x64,%al
     3b2:       3c ff                   cmp    $0xff,%al
     3b4:       75 04                   jne    0x3ba
     3b6:       66 49                   dec    %ecx
     3b8:       74 12                   je     0x3cc
     3ba:       a8 01                   test   $0x1,%al
     3bc:       74 06                   je     0x3c4
     3be:       e6 80                   out    %al,$0x80
     3c0:       e4 60                   in     $0x60,%al
     3c2:       eb 04                   jmp    0x3c8
     3c4:       a8 02                   test   $0x2,%al
     3c6:       74 0a                   je     0x3d2
     3c8:       66 4a                   dec    %edx
     3ca:       75 e2                   jne    0x3ae
     3cc:       66 83 c8 ff             or     $0xffffffff,%eax
     3d0:       66 c3                   retl
     3d2:       66 31 c0                xor    %eax,%eax
     3d5:       66 c3                   retl
                                                       a20_test
     3d7:       66 53                   push   %ebx
     3d9:       66 31 d2                xor    %edx,%edx
     3dc:       8e e2                   mov    %dx,%fs
     3de:       66 83 ca ff             or     $0xffffffff,%edx
     3e2:       8e ea                   mov    %dx,%gs
     3e4:       64 66 8b 0e 00 02       mov    %fs:0x200,%ecx
     3ea:       66 89 ca                mov    %ecx,%edx
     3ed:       67 66 8d 1c 01          lea    (%eax,%eax,1),%ebx
     3f2:       eb 14                   jmp    0x408
     3f4:       66 42                   inc    %edx
     3f6:       64 66 89 16 00 02       mov    %edx,%fs:0x200
     3fc:       e6 80                   out    %al,$0x80
     3fe:       65 66 a1 10 02          mov    %gs:0x210,%eax
     403:       66 31 d0                xor    %edx,%eax
     406:       75 08                   jne    0x410
     408:       66 39 da                cmp    %ebx,%edx
     40b:       75 e7                   jne    0x3f4
     40d:       66 31 c0                xor    %eax,%eax
     410:       64 66 89 0e 00 02       mov    %ecx,%fs:0x200
     416:       66 5b                   pop    %ebx
     418:       66 c3                   retl
                                                       enable_a20
     41a:       66 56                   push   %esi
     41c:       66 53                   push   %ebx
     41e:       66 83 ec 34             sub    $0x34,%esp
     422:       66 bb 00 01 00 00       mov    $0x100,%ebx
     428:       e9 c6 00                jmp    0x4f1
     42b:       66 b8 20 00 00 00       mov    $0x20,%eax
     431:       66 e8 a0 ff ff ff       calll  0x3d7
     437:       66 85 c0                test   %eax,%eax
```

```
43a:       74 06                    je     0x442
43c:       66 31 c0                 xor    %eax,%eax
43f:       e9 b9 00                 jmp    0x4fb
442:       67 66 8d 44 24 04        lea    0x4(%eax,%eax,1),%eax
448:       66 e8 c2 14 00 00        calll  0x1910
44e:       67 c7 44 24 20 01 24     movw   $0x2401,0x20(%eax,%eax,1)
455:       66 31 c9                 xor    %ecx,%ecx
458:       67 66 8d 54 24 04        lea    0x4(%eax,%eax,1),%edx
45e:       66 b8 15 00 00 00        mov    $0x15,%eax
464:       66 e8 9e 00 00 00        calll  0x508
46a:       66 b8 20 00 00 00        mov    $0x20,%eax
470:       66 e8 61 ff ff ff        calll  0x3d7
476:       66 85 c0                 test   %eax,%eax
479:       75 c1                    jne    0x43c
47b:       66 e8 1f ff ff ff        calll  0x3a0
481:       66 89 c6                 mov    %eax,%esi
484:       66 b8 20 00 00 00        mov    $0x20,%eax
48a:       66 e8 47 ff ff ff        calll  0x3d7
490:       66 85 c0                 test   %eax,%eax
493:       75 a7                    jne    0x43c
495:       66 85 f6                 test   %esi,%esi
498:       74 1f                    je     0x4b9
49a:       e4 92                    in     $0x92,%al
49c:       66 83 e0 fe              and    $0xfffffffe,%eax
4a0:       66 83 c8 02              or     $0x2,%eax
4a4:       e6 92                    out    %al,$0x92
4a6:       66 b8 00 00 20 00        mov    $0x200000,%eax
4ac:       66 e8 25 ff ff ff        calll  0x3d7
4b2:       66 85 c0                 test   %eax,%eax
4b5:       74 3a                    je     0x4f1
4b7:       eb 83                    jmp    0x43c
4b9:       66 e8 e1 fe ff ff        calll  0x3a0
4bf:       b0 d1                    mov    $0xd1,%al
4c1:       e6 64                    out    %al,$0x64
4c3:       66 e8 d7 fe ff ff        calll  0x3a0
4c9:       b0 df                    mov    $0xdf,%al
4cb:       e6 60                    out    %al,$0x60
4cd:       66 e8 cd fe ff ff        calll  0x3a0
4d3:       b0 ff                    mov    $0xff,%al
4d5:       e6 64                    out    %al,$0x64
4d7:       66 e8 c3 fe ff ff        calll  0x3a0
4dd:       66 b8 00 00 20 00        mov    $0x200000,%eax
4e3:       66 e8 ee fe ff ff        calll  0x3d7
4e9:       66 85 c0                 test   %eax,%eax
4ec:       74 ac                    je     0x49a
4ee:       e9 4b ff                 jmp    0x43c
4f1:       66 4b                    dec    %ebx
4f3:       0f 85 34 ff              jne    0x42b
4f7:       66 83 c8 ff              or     $0xffffffff,%eax
4fb:       66 83 c4 34              add    $0x34,%esp
4ff:       66 5b                    pop    %ebx
501:       66 5e                    pop    %esi
503:       66 c3                    retl
```

— Le coprocesseur arithmétique est réinitialisé (lignes 115 et 116, renvoyant à la fonction définie aux instructions d'adresses `0x11e8` à `0x11f1`, soit lignes 46 à 55).

— Toutes les interruptions du PIC sont inhibées (lignes 117 et 118, renvoyant à la fonction définie par les instructions d'adresses `0x11f3` à `0x11fd`, soit lignes 35 à 44).

— La table des descripteurs d'interruption (en mode protégé) est initialisée (lignes 120 et 121, renvoyant à la fonction définie à l'instruction d'adresse `0x11ff` soit lignes 92 à 99) au seul descripteur nul.

— La table globale des descripteurs est initialisée (ligne 122 renvoyant à la fonction définie lignes 57 à 90) : un descripteur de segment de code, un descripteur de segment de données et un descripteur de TSS.

— Le passage au mode protégé est réalisé en appelant (instructions d'adresses `0x1227` à `0x1232`, soit lignes 123 et 124) la routine `protected_mode_jump()`.

14.3.4 La routine `protected_mode_jump()`

La routine `protected_mode_jump()` est définie dans le fichier `arch/x86/boot/pmjump.S` et commence à partir de l'instruction d'adresse `0x1238` dans le code, comme nous venons de le voir :

```
$ objdump -D -b binary -m i8086 -Maddr16,data16 --start-address=0x1238
--stop-address=0x1261 /boot/vmlinuz64-3.4.42
/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:

000011b0 <.data+0x1238>:
                                                    |11 /*
                                                    |12  * The actual transition into protected mode
                                                    |13  */
                                                    |
                                                    |19
                                                    |20      .text
                                                    |21      .code16
                                                    |22
                                                    |23 /*
                                                    |24  * void protected_mode_jump(u32 entrypoint,
                                                    |                              u32 bootparams);
                                                    |25  */
                                                    |26 GLOBAL(protected_mode_jump)
    1238:  66 89 d6                mov    %edx,%esi |27      movl    %edx, %esi              # Pointer to
                                                    |        boot_params table
                                                    |28
    123b:  66 31 db                xor    %ebx,%ebx |29      xorl    %ebx, %ebx
    123e:  8c cb                   mov    %cs,%bx   |30      movw    %cs, %bx
    1240:  66 c1 e3 04             shl    $0x4,%ebx |31      shll    $4, %ebx
    1244:  66 01 1e 5c 12          add    %ebx,0x125c |32     addl    %ebx, 2f
    1249:  eb 00                   jmp    0x124b    |33      jmp     1f                      # Short jump to
                                                    |        serialize on 386/486
                                                    |34 1:
                                                    |35
    124b:  b9 18 00                mov    $0x18,%cx |36      movw    $__BOOT_DS, %cx
    124e:  bf 20 00                mov    $0x20,%di |37      movw    $__BOOT_TSS, %di
                                                    |38
    1251:  0f 20 c2                mov    %cr0,%edx |39      movl    %cr0, %edx
    1254:  80 ca 01                or     $0x1,%dl  |40      orb     $X86_CR0_PE, %dl        # Protected
                                                    |                                        mode
    1257:  0f 22 c2                mov    %edx,%cr0 |41      movl    %edx, %cr0
                                                    |42
                                                    |43      # Transition to 32-bit mode
    125a:  66 ea c7 2e 00 00 10    ljmpl  $0x10,$0x2ec7 |44     .byte   0x66, 0xea              # ljmpl opcode
                                                    |45 2:   .long   in_pm32                 # offset
                                                    |46      .word   __BOOT_CS               # segment
    1261:  00                                       |
                                                    |47 ENDPROC(protected_mode_jump)
                                                    |48
```

— On fait pointer le registre `ESI` sur la table des paramètres de démarrage, dont l'adresse a été passée en paramètre (instruction d'adresse `0x1238`, soit ligne 27).

— On place l'adresse physique du segment de code (en mode réel) à l'étiquette `in_pm32` (instructions d'adresses `0x123b` à `0x1244`, soit lignes 29 à 32 et 45).

— On effectue un saut court à la ligne suivante (instruction d'adresse `0x1249`, soit lignes 33 et 34).

— On initialise les registres `CX` (instruction d'adresse `0x124b`, soit ligne 36) et `DI` (instruction d'adresse `0x124e`, soit ligne 37).

— On passe au mode protégé en changeant la valeur du registre `CR0` (instructions d'adresses `0x1251` à `0x1257`, soit lignes 39 à 41).

— On effectue un saut long pour initialiser la valeur du registre de code `CS` en mode protégé (instructions d'adresses `0x125a` et `0x1261`, soit lignes 43 à 46).

14.3.5 Initialisation de Linux en mode protégé

Nous venons donc de passer au mode protégé. Reste à réaliser un certain nombre d'initialisations. Nous venons d'effectuer un saut à l'instruction `0x2ec7`, dont le source se trouve toujours dans le fichier `arch/x86/boot/pmjump.S`. Puisque nous sommes maintenant en mode 32 bits (d'après le choix du sélecteur du segment de code), nous devons désassembler avec un paramètre différent :

```
$objdump -D -b binary -mi386 --start-address=0x2ec7 /boot/vmlinuz64-3.4.42

/boot/vmlinuz64-3.4.42:     file format binary

Disassembly of section .data:
2ec7 <.data+0x2ec7>:
                                        |49      .code32
                                        |50      .section ".text32","ax"
                                        |51 GLOBAL(in_pm32)
                                        |52      # Set up data segments for flat 32-bit mode
2ec7:     8e d9          mov    %ecx,%ds  |53      movl    %ecx, %ds
2ec9:     8e c1          mov    %ecx,%es  |54      movl    %ecx, %es
2ecb:     8e e1          mov    %ecx,%fs  |55      movl    %ecx, %fs
2ecd:     8e e9          mov    %ecx,%gs  |56      movl    %ecx, %gs
2ecf:     8e d1          mov    %ecx,%ss  |57      movl    %ecx, %ss
                                        |58      # The 32-bit code sets up its own stack, but this way we
                                        |          do have
                                        |59      # a valid stack if some debugging hack wants to use it.
2ed1:     01 dc          add    %ebx,%esp |60      addl    %ebx, %esp
                                        |61
                                        |62      # Set up TR to make Intel VT happy
2ed3:     0f 00 df       ltr    %di       |63      ltr     %di
                                        |64
                                        |65      # Clear registers to allow for future extensions to the
                                        |66      # 32-bit boot protocol
2ed6:     31 c9          xor    %ecx,%ecx |67      xorl    %ecx, %ecx
2ed8:     31 d2          xor    %edx,%edx |68      xorl    %edx, %edx
2eda:     31 db          xor    %ebx,%ebx |69      xorl    %ebx, %ebx
2edc:     31 ed          xor    %ebp,%ebp |70      xorl    %ebp, %ebp
2ede:     31 ff          xor    %edi,%edi |71      xorl    %edi, %edi
                                        |72
                                        |73      # Set up LDTR to make Intel VT happy
2ee0:     0f 00 d1       lldt   %cx       |74      lldt    %cx
                                        |75
2ee3:     ff e0          jmp    *%eax     |76      jmpl    *%eax                   # Jump to the 32-bit
                                        | entrypoint
                                        |77 ENDPROC(in_pm32)
```

— On se trouve en mode d'instructions 32 bits (vu le descripteur de code choisi), d'où la directive de la ligne 49.

— On initialise les registres de segment `DS` à `GS`, ainsi que le registre de pile `SS`, avec le même sélecteur que celui du segment des données (instructions d'adresses `0x124b` et `0x2ec7` à `0x2ecf`, soit lignes 36 et 52 à 57).

— On initialise momentanément le pointeur de pile à sa valeur pour le mode réel en ajoutant à la valeur en cours l'ancienne valeur du registre de segment `CS` (instructions d'adresses `0x1244`, soit ligne 32, et `0x2ed1`, soit lignes 58 à 60).

— On initialise le registre de tâche, bien que cela ne serve à rien pour l'instant (instruction d'adresse `0x2ed3`, soit lignes 62 et 63).

— On initialise les registres `ECX`, `EDX`, `EBX`, `EBP` et `EDI` à zéro (instructions d'adresses `0x2ed6` à `0x2ede`, soit lignes 65 à 71).

— On initialise le registre de la table locale de descripteurs, bien que cela ne serve à rien pour l'instant (instruction d'adresse `0x2ee0`, soit lignes 73 et 74).

— Maintenant que nous sommes en mode protégé, on saute au point d'entrée du mode 32 bits (instruction d'adresse `0x2ee3`, soit ligne 76), transmis, ligne 123 du fichier `pm.c`, en paramètre de la fonction `protected_mode_jump()`.

L'adresse est le contenu de `CS:40a4` (voir ligne de l'instruction d'adresse `0x122e`). Rappelons que nous avons copié les 115 octets du fichier, à partir du décalage `0x1f1` vers la mémoire vive en `0:4081`. L'adresse `CS:40a4` correspond donc à l'emplacement `0x214` du fichier. On y trouve `0x00 00 10 00`, soit `0x100000`.

Ceci correspond au décalage `0x3C00` du code ? ? ?

14.4 Passage au mode 64 bits

14.4.1 Routine de passage au mode 64 bits

Fichier source.- Le début du code 32 bits se trouve, suivant l'option choisie pour la compilation (architecture x86 ou x86-64), dans le fichier `arch/x86/boot/compressed/head_32.S` ou dans le fichier `arch/x86/boot/compressed/head_64.S`. C'est le deuxième, comprenant 389 lignes, qui nous intéresse ici.

Emplacement dans le fichier de code.- Le raisonnement ci-dessus n'est pas très probant pour déterminer le décalage dans le fichier de code. Essayons donc de le déterminer autrement.

Plus généralement, étant donnés :

(1) un fichier exécutable '`foo`' et

(2) un fichier source en langage C (`bar.c`) ou en langage d'assemblage (`bar.S`) d'une partie (c'est important !) de '`foo`',

voyons une façon de trouver le décalage dans le fichier '`foo`' correspondant au début de '`bar`'.

Ceci exige une condition : que toutes les traces de la compilation de '`foo`' soient disponibles, c'est-à-dire que l'on puisse disposer du fichier '`bar.o`'.

Considérons donc '`bar.o`'. La commande « `file bar.o` » nous montre son format, ce qui nous permet de déterminer les paramètres nécessaires de '`objdump`'. Connaissant l'architecture, désassemblons '`bar.o`' et choisissons une suite d'octets dont l'occurrence dans '`foo`' nous permettra de repérer l'emplacement de '`bar`' dans '`foo`'. Pour vérifier qu'il s'agit bien du bon emplacement, il suffit de désassembler '`foo`' à partir du décalage trouvé.

Dans notre cas, le fichier '`foo`' est `/boot/vmlinuz64-3.4.42` et le fichier '`bar.S`' qui nous intéresse est `/usr/src/linux/arch/x86/boot/compressed/head_64.S`. Désassemblons ce fichier `head_64.o` :

```
$ objdump -d /usr/src/linux/arch/x86/boot/compressed/head_64.o

/usr/src/linux/arch/x86/boot/compressed/head_64.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <relocated>:
0:  31 c0                   xor %eax,%eax
2:  48 8d 3d 00 00 00 00    lea 0x0(%rip),%rdi # 9  <relocated+0x9>
9:  48 8d 0d 00 00 00 00    lea 0x0(%rip),%rcx # 10 <relocated+0x10>
10: 48 29 f9                sub %rdi,%rcx
13: 48 c1 e9 03             shr $0x3,%rcx
17: f3 48 ab                rep stos %rax,%es:(%rdi)
1a: 48 8d 15 00 00 00 00    lea 0x0(%rip),%rdx # 21 <relocated+0x21>
21: 48 8d 0d 00 00 00 00    lea 0x0(%rip),%rcx # 28 <relocated+0x28>
28: 48 39 ca                cmp %rcx,%rdx
2b: 73 09                   jae 36 <relocated+0x36>
2d: 48 01 1a                add %rbx,(%rdx)
30: 48 83 c2 08             add $0x8,%rdx
34: eb f2                   jmp 28 <relocated+0x28>
36: 56                      push %rsi
37: 48 89 f7                mov %rsi,%rdi
3a: 48 8d 35 00 00 00 00    lea 0x0(%rip),%rsi # 41 <relocated+0x41>
41: 48 8d 15 00 00 00 00    lea 0x0(%rip),%rdx # 48 <relocated+0x48>
```

```
48: b9 00 00 00 00          mov $0x0,%ecx
4d: 49 89 e8                mov %rbp,%r8
50: e8 00 00 00 00          callq 55 <relocated+0x55>
55: 5e                      pop %rsi
56: ff e5                   jmpq *%rbp
```

C'est suffisant pour le moment. Cherchons l'occurrence de '31 c0 48 8d 3d' dans le fichier vmlinuz64-3.4.42. Attention ! il ne faut pas inclure '00 00 00 00' dans le motif.

```
$ od --format=x1 /boot/vmlinuz64-3.4.42 |grep '31 c0 48 8d 3d'
24016760 31 c0 48 8d 3d 07 4e 00 00 48 8d 0d 58 0e 01 00
```

On trouve une seule occurence. Tant mieux ! N'oublions pas que od affiche les adresses en octal : $(24016760)_8$ = 0x501df0. Voyons :

```
$ objdump -D -b binary -m i386:x86-64 --start-address=0x501df0
/boot/vmlinuz64-3.4.42 |less

/boot/vmlinuz64-3.4.42: file format binary

Disassembly of section .data:

0000000000501df0 <.data+0x501df0>:
501df0: 31 c0               xor %eax,%eax
501df2: 48 8d 3d 07 4e 00 00 lea 0x4e07(%rip),%rdi
# 0x506c00
501df9: 48 8d 0d 58 0e 01 00 lea 0x10e58(%rip),%rcx
# 0x512c58
501e00: 48 29 f9            sub %rdi,%rcx
501e03: 48 c1 e9 03         shr $0x3,%rcx
501e07: f3 48 ab            rep stos %rax,%es:(%rdi)
501e0a: 48 8d 15 67 4d 00 00 lea 0x4d67(%rip),%rdx
# 0x506b78
501e11: 48 8d 0d 88 4d 00 00 lea 0x4d88(%rip),%rcx
# 0x506ba0
501e18: 48 39 ca            cmp %rcx,%rdx
501e1b: 73 09               jae 0x501e26
501e1d: 48 01 1a            add %rbx,(%rdx)
501e20: 48 83 c2 08         add $0x8,%rdx
501e24: eb f2               jmp 0x501e18
501e26: 56                  push %rsi
501e27: 48 89 f7            mov %rsi,%rdi
501e2a: 48 8d 35 cf 4d 00 00 lea 0x4dcf(%rip),%rsi
# 0x506c00
501e31: 48 8d 15 20 20 b0 ff lea -0x4fdfe0(%rip),%rdx
# 0x3e58
501e38: b9 98 df 4f 00      mov $0x4fdf98,%ecx
501e3d: 49 89 e8            mov %rbp,%r8
501e40: e8 8b 2d 00 00      callq 0x504bd0
501e45: 5e                  pop %rsi
501e46: ff e5               jmpq *%rbp
```

C'est exactement ce que nous cherchions !

Analyse.- Nous pouvons donc continuer à analyser le code :

```
/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:

00003c00 <.data+0x3c00>:
                                                        |7   /*
                                                        |8    *  head.S contains the 32-bit startup
                                                        |       code.
                                                        |9    *
                                                        |10   * NOTE!!! Startup happens at absolute
                                                        |       address 0x00001000, which is also where
                                                        |11   * the page directory will exist. The
                                                        |       startup code will be overwritten by
                                                        |12   * the page directory. [According to
                                                        |       comments etc elsewhere on a compressed
                                                        |13   * kernel it will end up at 0x1000 + 1Mb I
                                                        |       hope so as I assume this. - AC]
                                                        |14   *
                                                        |15   * Page 0 is deliberately kept safe, since
                                                        |       System Management Mode code in
                                                        |16   * laptops may need to access the BIOS data
                                                        |       stored there.  This is also
                                                        |17   * useful for future device drivers that
                                                        |       either access the BIOS via VM86
                                                        |18   * mode.
                                                        |19   */
                                                        |20
                                                        |21  /*
                                                        |22   * High loaded stuff by Hans Lermen &
                                                        |       Werner Almesberger, Feb. 1996
                                                        |23   */
                                                        |24      .code32
                                                        |25      .text
                                                        |26
                                                        |
                                                        |37      __HEAD
                                                        |38      .code32
                                                        |39  ENTRY(startup_32)
    3c00:       fc                      cld             |40      cld
                                                        |41      /*
                                                        |42       * Test KEEP_SEGMENTS flag to see if
                                                        |           the bootloader is asking
                                                        |43       * us to not reload segments
                                                        |44       */
    3c01:       f6 86 11 02 00 00 40    testb  $0x40,0x211(%esi)|45      testb $(1<<6), BP_loadflags(%esi)
    3c08:       75 0c                   jne    0x3c16           |46      jnz 1f
                                                        |47
    3c0a:       fa                      cli             |48      cli
    3c0b:       b8 18 00 00 00          mov    $0x18,%eax       |49      movl    $(__KERNEL_DS), %eax
    3c10:       8e d8                   mov    %eax,%ds         |50      movl    %eax, %ds
    3c12:       8e c0                   mov    %eax,%es         |51      movl    %eax, %es
    3c14:       8e d0                   mov    %eax,%ss         |52      movl    %eax, %ss
                                                        |53  1:
                                                        |54
                                                        |55  /*
                                                        |56   * Calculate the delta between where we
                                                        |       were compiled to run
                                                        |57   * at and where we were actually loaded at.
                                                        |       This can only be done
                                                        |58   * with a short local call on x86.  Nothing
                                                        |       else will tell us what
                                                        |59   * address we are running at.  The reserved
                                                        |       chunk of the real-mode
                                                        |60   * data at 0x1e4 (defined as a scratch
                                                        |       field) are used as the stack
                                                        |61   * for this calculation. Only 4 bytes are
                                                        |       needed.
                                                        |62   */
    3c16:       8d a6 e8 01 00 00       lea    0x1e8(%esi),%esp |63      leal    (BP_scratch+4)(%esi), %esp
    3c1c:       e8 00 00 00 00          call   0x3c21           |64      call    1f
    3c21:       5d                      pop    %ebp             |65  1:  popl    %ebp
```

```
3c22:       81 ed 21 00 00 00       sub    $0x21,%ebp       |66      subl    $1b, %ebp
                                                            |67
                                                            |68  /* setup a stack and make sure cpu supports
                                                            |       long mode. */
3c28:       b8 00 f0 50 00          mov    $0x50f000,%eax   |69      movl    $boot_stack_end, %eax
3c2d:       01 e8                   add    %ebp,%eax        |70      addl    %ebp, %eax
3c2f:       89 c4                   mov    %eax,%esp        |71      movl    %eax, %esp
                                                            |72
3c31:       e8 b1 00 00 00          call   0x3ce7           |73      call    verify_cpu
3c36:       85 c0                   test   %eax,%eax        |74      testl   %eax, %eax
3c38:       0f 85 a6 00 00 00       jne    0x3ce4           |75      jnz     no_longmode
                                                            |76
                                                            |77  /*
                                                            |78   * Compute the delta between where we were
                                                            |       compiled to run at
                                                            |79   * and where the code will actually run at.
                                                            |80   *
                                                            |81   * %ebp contains the address we are loaded
                                                            |       at by the boot loader and %ebx
                                                            |82   * contains the address where we should
                                                            |       move the kernel image temporarily
                                                            |83   * for safe in-place decompression.
                                                            |84   */
                                                            |85
                                                            |86  #ifdef CONFIG_RELOCATABLE
                                                            |
                                                            |93  #else
3c3e:       bb 00 00 00 01          mov    $0x1000000,%ebx  |94      movl    $LOAD_PHYSICAL_ADDR, %ebx
                                                            |95  #endif
                                                            |96
                                                            |97      /* Target address to relocate to for
                                                            |           decompression */
3c43:       81 c3 00 d0 99 00       add    $0x99d000,%ebx   |98      addl    $z_extract_offset, %ebx
                                                            |99
                                                            |100 /*
                                                            |101  * Prepare for entering 64 bit mode
                                                            |102  */
                                                            |103
                                                            |104     /* Load new GDT with the 64bit segments
                                                            |           using 32bit descriptor */
3c49:       8d 85 a0 2f 50 00       lea    0x502fa0(%ebp),%eax |105     leal    gdt(%ebp), %eax
3c4f:       89 85 a2 2f 50 00       mov    %eax,0x502fa2(%ebp) |106     movl    %eax, gdt+2(%ebp)
3c55:       0f 01 95 a0 2f 50 00    lgdtl  0x502fa0(%ebp)   |107     lgdt    gdt(%ebp)
                                                            |108
                                                            |109     /* Enable PAE mode */
3c5c:       b8 20 00 00 00          mov    $0x20,%eax       |110     movl    $(X86_CR4_PAE), %eax
3c61:       0f 22 e0                mov    %eax,%cr4        |111     movl    %eax, %cr4
                                                            |112
                                                            |113  /*
                                                            |114   * Build early 4G boot pagetable
                                                            |115   */
                                                            |116     /* Initialize Page tables to 0 */
3c64:       8d bb 00 00 51 00       lea    0x510000(%ebx),%edi |117     leal    pgtable(%ebx), %edi
3c6a:       31 c0                   xor    %eax,%eax        |118     xorl    %eax, %eax
3c6c:       b9 00 18 00 00          mov    $0x1800,%ecx     |119     movl    $((4096*6)/4), %ecx
3c71:       f3 ab                   rep stos %eax,%es:(%edi)|120     rep     stosl
                                                            |121
                                                            |122     /* Build Level 4 */
3c73:       8d bb 00 00 51 00       lea    0x510000(%ebx),%edi |123     leal    pgtable + 0(%ebx), %edi
3c79:       8d 87 07 10 00 00       lea    0x1007(%edi),%eax|124     leal    0x1007 (%edi), %eax
3c7f:       89 07                   mov    %eax,(%edi)      |125     movl    %eax, 0(%edi)
                                                            |126
                                                            |127     /* Build Level 3 */
3c81:       8d bb 00 10 51 00       lea    0x511000(%ebx),%edi |128     leal    pgtable + 0x1000(%ebx), %edi
3c87:       8d 87 07 10 00 00       lea    0x1007(%edi),%eax|129     leal    0x1007(%edi), %eax
3c8d:       b9 04 00 00 00          mov    $0x4,%ecx        |130     movl    $4, %ecx
3c92:       89 07                   mov    %eax,(%edi)      |131 1:  movl    %eax, 0x00(%edi)
3c94:       05 00 10 00 00          add    $0x1000,%eax     |132     addl    $0x00001000, %eax
3c99:       83 c7 08                add    $0x8,%edi        |133     addl    $8, %edi
3c9c:       49                      dec    %ecx             |134     decl    %ecx
3c9d:       75 f3                   jne    0x3c92           |135     jnz     1b
                                                            |136
                                                            |137     /* Build Level 2 */
3c9f:       8d bb 00 20 51 00       lea    0x512000(%ebx),%edi |138     leal    pgtable + 0x2000(%ebx), %edi
```

```
3ca5:       b8 83 01 00 00          mov    $0x183,%eax      |139     movl    $0x00000183, %eax
3caa:       b9 00 08 00 00          mov    $0x800,%ecx      |140     movl    $2048, %ecx
3caf:       89 07                   mov    %eax,(%edi)      |141 1:  movl    %eax, 0(%edi)
3cb1:       05 00 00 20 00          add    $0x200000,%eax   |142     addl    $0x00200000, %eax
3cb6:       83 c7 08                add    $0x8,%edi        |143     addl    $8, %edi
3cb9:       49                      dec    %ecx             |144     decl    %ecx
3cba:       75 f3                   jne    0x3caf           |145     jnz     1b
                                                            |146
                                                            |147     /* Enable the boot page tables */
3cbc:       8d 83 00 00 51 00       lea    0x510000(%ebx),%eax |148    leal    pgtable(%ebx), %eax
3cc2:       0f 22 d8                mov    %eax,%cr3        |149     movl    %eax, %cr3
                                                            |150
                                                            |151     /* Enable Long mode in EFER (Extended
                                                            |           Feature Enable Register) */
3cc5:       b9 80 00 00 c0          mov    $0xc0000080,%ecx |152     movl    $MSR_EFER, %ecx
3cca:       0f 32                   rdmsr                   |153     rdmsr
3ccc:       0f ba e8 08             bts    $0x8,%eax        |154     btsl    $_EFER_LME, %eax
3cd0:       0f 30                   wrmsr                   |155     wrmsr
                                                            |156
                                                            |157     /*
                                                            |158      * Setup for the jump to 64bit mode
                                                            |159      *
                                                            |160      * When the jump is performend we will
                                                            |           be in long mode but
                                                            |161      * in 32bit compatibility mode with
                                                            |           EFER.LME = 1, CS.L = 0, CS.D = 1
                                                            |162      * (and in turn EFER.LMA = 1). To jump
                                                            |           into 64bit mode we use
                                                            |163      * the new gdt/idt that has __KERNEL_CS
                                                            |           with CS.L = 1.
                                                            |164      * We place all of the values on our
                                                            |           mini stack so lret can
                                                            |165      * used to perform that far jump.
                                                            |166      */
3cd2:       6a 10                   push   $0x10            |167     pushl   $__KERNEL_CS
3cd4:       8d 85 00 02 00 00       lea    0x200(%ebp),%eax |168     leal    startup_64(%ebp), %eax
3cda:       50                      push   %eax             |169     pushl   %eax
                                                            |170
                                                            |171     /* Enter paged protected Mode,
                                                            |           activating Long Mode */
3cdb:       b8 01 00 00 80          mov    $0x80000001,%eax |172     movl    $(X86_CR0_PG | X86_CR0_PE),
                                                            |                 %eax
                                                            |         /* Enable Paging and Protected mode */
3ce0:       0f 22 c0                mov    %eax,%cr0        |173     movl    %eax, %cr0
                                                            |174
                                                            |175     /* Jump from 32bit compatibility mode
                                                            |           into 64bit mode. */
3ce3:       cb                      lret                    |176     lret
                                                            |177 ENDPROC(startup_32)
                                                            |178
                                                            |179 no_longmode:
                                                            |180    /* This isn't an x86-64 CPU so hang */
                                                            |181 1:
                                                            |182    hlt
                                                            |183    jmp     1b
                                                            |184
                                                            |
                                                            |341
                                                            |342    .data
                                                            |343 gdt:
                                                            |344    .word gdt_end - gdt
                                                            |345    .long gdt
                                                            |346    .word 0
                                                            |347    .quad 0x0000000000000000
                                                            |       /* NULL descriptor */
                                                            |348    .quad 0x00af9a000000ffff
                                                            |       /* __KERNEL_CS */
                                                            |349    .quad 0x00cf92000000ffff
                                                            |       /* __KERNEL_DS */
                                                            |350    .quad 0x0080890000000000
                                                            |       /* TS descriptor */
                                                            |351    .quad   0x0000000000000000
                                                            |       /* TS continued */
                                                            |352 gdt_end:
```

```
|353
|
|364
|365 /*
|366  * Space for page tables (not in .bss so
|       not zeroed)
|367  */
|368    .section ".pgtable","a",@nobits
|369    .balign 4096
|370 pgtable:
|371    .fill 6*4096, 1, 0
|372
```

— Il est dommage que le commentaire de début n'ait pas changé : la dernière date que nous voyons apparaître est 1996 (ligne 22), donc avant l'apparition de l'architecture x86-64.

— Le commentaire de la ligne 8 nous rappelle qu'il s'agit du point d'entrée, une fois le passage au mode protégé effectué. Celui de la ligne 10 nous rappelle que ce code a été placé par l'utilitaire de démarrage multiple à l'adresse `0x1000`.

— Nous sommes maintenant en mode d'instructions 32 bits, l'assembleur doit en tenir compte (ligne 24, répété ligne 38).

— On progressera en avant lors des opérations sur les chaînes de caractères (instruction d'adresse `0x3c00`, soit ligne 40), on inhibe les interruptions masquables (instruction d'adresse `0x3c0a`, soit ligne 48), on initialise les registres de segment `DS`, `ES` et `SS` avec le sélecteur du segment des données choisi (instructions d'adresses `0x3c0b` à `0x3c14`, soit lignes 49 à 52) et on initialise la pile (instructions d'adresses `0x3c28` à `0x3c2f`, soit lignes 68 à 71).

— On vérifie que le microprocesseur permet le mode long (instructions d'adresses `0x3c31` à `0x3c38`, soit lignes 73 à 75). Si ce n'est pas le cas, on s'arrête (lignes 179 à 183).

— Sinon on charge la table globale des descripteurs pour le mode 64 bits (instructions d'adresses `0x3c49` à `0x3c55`, soit lignes 104 à 107). Cette table se trouve lignes 343 à 352. Elle comprend le descripteur nul, un descripteur de segment de code, un descripteur de segment de données et un descripteur de tâche.

— On active le mode *Pagination étendue* (instructions d'adresses `0x3c5c` et `0x3c61`, soit lignes 109 à 111).

— On construit la table des pages pour les quatre premiers GiO (lignes 113 à 145) : les tables de pages sont toutes initialisées à zéro (instructions d'adresses `0x3c64` à `0x3c71`, soit lignes 116 à 120) ; la table de niveau 4 est placée à l'adresse `0x1000`, remplaçant le code de démarrage (instructions d'adresses `0x3c73` à `0x3c7f`, soit lignes 122 à 125, voir le commentaire des lignes 10 à 13) ; les quatre tables de niveau 3 sont placées ensuite (instructions d'adresses `0x3C81` à `0x3c9d`, soit lignes 127 à 135) ; les 2 048 tables de niveau 2 sont placées ensuite (instructions d'adresses `0x3c9f` à `0x3cba`, soit lignes 137 à 145).

— On active la pagination (instructions d'adresses `0x3cbc` à `0x3cc2`, soit lignes 147 à 149).

— On active le mode long (instructions d'adresses `0x3cc5` à `0x3cd0`, soit lignes 151 à 155).

— On effectue un saut long pour passer au mode long, de façon à ce que le registre `CS` soit changé (instructions d'adresses `0x3cd2` à `0x3cda`, soit lignes 157 à 169). Nous sommes alors dans le mode de compatibilité.

— On change le contenu du registre `CR0` (instructions d'adresses `0x3cdb` à `0x3ce0`, soit lignes 171 à 173) et on effectue un saut (instruction d'adresse `0x3ce3`, soit lignes 175 et 176) pour passer dans le mode 64 bits .

14.4.2 Initialisation de Linux en mode 64 bits

Pour la suite, toujours dans le même fichier source, nous devons désassembler du code 64 bits, d'où l'utilisation d'une option nouvelle de `objdump` :

```
$ objdump -D -b binary -m i386:x86-64 --start-address=0x3e00 --stop-address=0x3e17 /boot/vmlinuz64-3.4.42

/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:

00003e00 <.data+0x3e00>:
                                                    |187  /*
                                                    |188   * Be careful here startup_64 needs to be at a
                                                    |        predictable
                                                    |189   * address so I can export it in an ELF header.
                                                    |        Bootloaders
                                                    |190   * should look at the ELF header to find this
                                                    |        address, as
                                                    |191   * it may change in the future.
                                                    |192   */
                                                    |193  .code64
                                                    |194  .org 0x200
                                                    |195 ENTRY(startup_64)
                                                    |196 /*
                                                    |197  * We come here either from startup_32
                                                    |        or directly from a
                                                    |198  * 64bit bootloader.  If we come here from a
                                                    |        bootloader we depend on
                                                    |199  * an identity mapped page table being provied
                                                    |        that maps our
                                                    |200  * entire text+data+bss and hopefully all of
                                                    |        memory.
                                                    |201  */
                                                    |
                                                    |232
                                                    |233 /* Setup data segments. */
    3e00:       31 c0                   xor    %eax,%eax   |234 xorl    %eax, %eax
    3e02:       8e d8                   mov    %eax,%ds    |235 movl    %eax, %ds
    3e04:       8e c0                   mov    %eax,%es    |236 movl    %eax, %es
    3e06:       8e d0                   mov    %eax,%ss    |237 movl    %eax, %ss
    3e08:       8e e0                   mov    %eax,%fs    |238 movl    %eax, %fs
    3e0a:       8e e8                   mov    %eax,%gs    |239 movl    %eax, %gs
    3e0c:       0f 00 d0                lldt   %ax         |240 lldt    %ax
    3e0f:       b8 20 00 00 00          mov    $0x20,%eax  |241 movl    $0x20, %eax
    3e14:       0f 00 d8                ltr    %ax         |242 ltr     %ax
                                                    |243
                                                    |244  /*
                                                    |245   * Compute the decompressed kernel start
                                                    |        address.  It is where
                                                    |246   * we were loaded at aligned to a 2M boundary.
                                                    |        %rbp contains the
                                                    |247   * decompressed kernel start address.
                                                    |248   *
                                                    |249   * If it is a relocatable kernel then decompress
                                                    |        and run the kernel
                                                    |250   * from load address aligned to 2MB addr,
                                                    |        otherwise decompress and
                                                    |251   * run the kernel from LOAD_PHYSICAL_ADDR
                                                    |252   *
                                                    |253   * We cannot rely on the calculation done in
                                                    |        32-bit mode, since we
                                                    |254   * may have been invoked via the 64-bit entry
                                                    |        point.
                                                    |255   */
```

— L'assembleur doit traduire les instructions suivantes en code 64 bits, d'où la nécessité de la directive de la ligne 193.

— On place le code 64 bits en mémoire vive à partir de l'adresse `0x200`, que l'on soit arrivé ici à partir de `statup_32` (ce que nous venons de faire) ou directement *via* un chargeur EFI. Ceci est rappelé dans le commentaire des lignes 196 à 201.

— Nous pouvons oublier les lignes 202 à 231 du code source, qui ne concernent que le chargement EFI, ce qui n'est pas le cas dans notre exemple.

— On initialise les registres de segment de données `DS` à `GS` à 0, comme l'exige le mode 64 bits (instructions d'adresses `0x3e00` à `0x3e0a`, soit lignes 233 à 239).

— On initialise la table locale des descripteurs (instruction d'adresse `0x3e0c`, soit ligne 240), avec le descripteur nul (vu que `ax = 0`).

— On active la tâche système (instructions d'adresses `0x3e0f` et `0x3e14`, soit lignes 241 à 242).

— On décompresse ensuite l'image du noyau (voir le commentaire des lignes 244 à 255), ce qui ne nous intéresse pas ici.

```
$ objdump -D -b binary -m i386:x86-64 --start-address=0x3e17 --stop-address=0x3e58 /boot/vmlinuz64-3.4.42

/boot/vmlinuz64-3.4.42:     file format binary


Disassembly of section .data:

0000000000003e17 <.data+0x3e17>:
    3e17: 48 c7 c5 00 00 00 01  mov    $0x1000000,%rbp
    3e1e: 48 8d 9d 00 d0 99 00  lea    0x99d000(%rbp),%rbx
    3e25: 48 8d a3 00 f0 50 00  lea    0x50f000(%rbx),%rsp
    3e2c: 6a 00                 pushq  $0x0
    3e2e: 9d                    popfq
    3e2f: 56                    push   %rsi
    3e30: 48 8d 35 c1 2d 50 00  lea    0x502dc1(%rip),%rsi        # 0x506bf8
    3e37: 48 8d bb f8 2f 50 00  lea    0x502ff8(%rbx),%rdi
    3e3e: 48 c7 c1 00 30 50 00  mov    $0x503000,%rcx
    3e45: 48 c1 e9 03           shr    $0x3,%rcx
    3e49: fd                    std
    3e4a: f3 48 a5              rep movsq %ds:(%rsi),%es:(%rdi)
    3e4d: fc                    cld
    3e4e: 5e                    pop    %rsi
    3e4f: 48 8d 83 f0 e1 4f 00  lea    0x4fe1f0(%rbx),%rax
    3e56: ff e0                 jmpq   *%rax
```

```
1   /*
2    *  linux/boot/head.S
3    *
4    *  Copyright (C) 1991, 1992, 1993  Linus Torvalds
5    */
6
```

```
27  #include <linux/init.h>
28  #include <linux/linkage.h>
29  #include <asm/segment.h>
30  #include <asm/pgtable_types.h>
31  #include <asm/page_types.h>
32  #include <asm/boot.h>
33  #include <asm/msr.h>
34  #include <asm/processor-flags.h>
35  #include <asm/asm-offsets.h>
36
[...]
287
288 /*
289  * Copy the compressed kernel to the end of our buffer
290  * where decompression in place becomes safe.
291  */
[...]
301
302 /*
303  * Jump to the relocated address.
304  */
[...]
310
311 /*
```

```
312  * Clear BSS (stack is currently empty)
313  */
[...]
320
321 /*
322  * Adjust our own GOT
323  */
[...]
333
334 /*
335  * Do the decompression, and jump to the new kernel..
336  */
[...]
345
346 /*
347  * Jump to the decompressed kernel.
348  */
349     jmp     *%rbp
350
351     .code32
352 no_longmode:
353     /* This isn't an x86-64 CPU so hang */
354 1:
355     hlt
356     jmp     1b
357
358 #include "../../kernel/verify_cpu.S"
359
360     .data
361 gdt:
362     .word   gdt_end - gdt
363     .long   gdt
364     .word   0
365     .quad   0x0000000000000000      /* NULL descriptor */
366     .quad   0x00af9a000000ffff      /* __KERNEL_CS */
367     .quad   0x00cf92000000ffff      /* __KERNEL_DS */
368     .quad   0x0080890000000000      /* TS descriptor */
369     .quad   0x0000000000000000      /* TS continued */
370 gdt_end:
371
[...]
382
383 /*
384  * Space for page tables (not in .bss so not zeroed)
385  */
386     .section ".pgtable","a",@nobits
387     .balign 4096
388 pgtable:
389     .fill 6*4096, 1, 0
```

14.4.3 Cas d'un chargement EFI

Nous avons vu que Linux passe du mode réel au mode protégé, puis au mode long et, enfin, au mode 64 bits dans le cas d'un chargement utilisant le BIOS. Le BIOS, dont le principe date de 1981, a ses limites. Lors de son essai de conception d'un vrai microprocesseur 64 bits, appelé *Itanium*, *Intel*, contrairement à sa philosophie usuelle, n'a pas conservé la compatibilité ascendante avec le `8086`, ni même avec l'architecture IA-32. Il est vrai que ce microprocesseur n'a pas connu de succès, contrairement à l'architecture IA-32e d'AMD, et qu'il a été abandonné en janvier 2019.

Lorsque les premiers systèmes à base d'*Itanium* furent conçus, au milieu des années 1990, le BIOS fut remplacé par **EFI** (*Extensible Firmware Interface*), dont la spécification a été développée par *Intel*. Cette interface (ou **UEFI** pour *Unified Extensible Firmware Interface*) remplace, petit à petit, le BIOS, la plupart gardant la compatibilité avec le BIOS.

Lorsqu'on démarre avec EFI, on est déjà passé au mode 64 bits et Linux n'a pas besoin d'effectuer les passages indiqués ci-dessus.

14.5 Bibliographie

[Ceg-04] Patrick Cégielski, **Conception des systèmes d'exploitation : Le cas linux. Deuxième édition**, Eyrolles, XIII + 680 p., septembre 2004.