Linuxブートアップの仕組み

芳之内 弘

第1部　LILOブートアップ・ソースコードの解析

PCのリセット後に、Linuxが立ち上がるまでのシーケンスについて大まかな説明をしたものは、世の中に幾つか存在します(本稿末の参考文献を参照)。LILOの役割や動作、そして使い方は、それらの文献を参考にするのがベストです。しかしソースコードの中身まで立ち入って詳細に説明をしたものは見当たりません。そこでLinuxのブートアップの仕組みを詳しく理解するために、ブートローダとして多くのディストリビューションが採用している「LILO」(LInux LOader)のソースコードを逐一解析してみました。ブートアップの部分はハードウェアへの依存性が高く、またサイズの制限もあり、アセンブラで書かれています。ここではPC互換機(インテルi386系)のコードについて解析します。

PC互換機では、リセット後はBIOSが起動し、LILOもこのBIOSを使って作業をします。アセンブラやBIOSに関する書籍は現在入手難ですので、ソースコードで使われているものについては、これをやや詳しく説明しました。

第1部では、LILOのブートアップの部分である“first.S”と“second.S”のソースコードを詳しく解析します。

また、次号以降に掲載予定の第2部では、ブートアップの次の段階であるカーネルのソースに付属の“setup.S”についてソースコードの解析をします。さらに、マップインストーラとしてのLILO本体のソースコードを解析して、それを第3部としてお届けする予定です。この部分はC言語で書かれているので解析は比較的楽です。

LILOによるカーネルロードの仕組みの概要

LILOは、Linuxのみをロードするように設定されているものとして説明します。ソースコードを読むときには、シーケンスを理解していると作業が容易になりますので、以下、その概要を詳しく説明します。

1. PCがリセットされると、ファームウェア(ハードウェアに組み込まれたソフトウェア)のBIOSが立ち上がります。
2. BIOSはPCの初期設定を終了します。
3. BIOSは、ハードディスクの最初のセクタのプライマリブートローダを読み込み、これをメモリの‘0x7C00’に書き込みます。ただし、「システムコマンダー」などのマルチブートローダを使っている場合は、Linuxパーティションの最初のセクタにプライマリブートローダがあります。
4. このブートローダはLILOの一部分で、この512Bytesのコードはすぐに‘0x9A000’にコピーされます。次にプログラムは‘0x9A000’に実行が移され、スタックの先頭を“0x9B000”に設定します。
5. “L”を表示します。
6. 次に8セクタ(4096Bytes)のセカンダリブードローダを“0x9B000”に書き込みます。
7. 読み込みに失敗した場合は2桁のエラーコードを表示し、読み込みを再試行します。
8. 成功したときは‘I’を表示します。
9. セカンダリブートローダの‘0x9B000’にジャンプします。
10. パラメータで指定されている場合はシリアルポートの設定を行い‘LI’をポートに出力します。
11. セカンダリブートローダの先頭のデータをチェックして、エラーがあれば‘?’を無限に表示します。
12. “L”を表示します。
13. /boot/mapファイルからディスクリプタテーブルを2セクタ分読み込み、“0x9D200”に書き込みます。次にチェクサムを計算し、エラーの場合は‘-’を表示して停止します。
14. キーボード変換テーブルを1セクタ分読み込み、“0x9D800”に書き込みます。
15. デフォルトコマンドラインを1セクタ分読み込み、“0x9D600”に書き込みます。
16. デフォルトコマンドラインがイネーブルのときは、このブロックの最初の2Bytesであるマジックナンバー(0xF4F2)を‘0x6B6D’に書き換えて、元のセクタにオーバーライトします。

17. “0”を表示します。
18. プロンプトが設定されていると対話モードになります。
19. 改行して“boot：”を表示します。
20. キー入力を待ちます。入力がないときはタイムアウトを待ちます。
21. 起動するブートイメージが決定されます。デフォルトではlilo.confの最初のイメージが選択されます。
22. 「Loading ＜ブートイメージ名＞」を表示します。
23. 読み込まれるRAMディスクのファイルサイズが指定されている場合は、このファイル用のMAPテーブルを読み込みます。
24. このMAPテーブルに従ってRAMディスクファイル(initrd)を拡張メモリ(0x100000以降)に読み込みます。
25. 次にカーネルのMAPテーブルを読み込みます。
26. このMAPテーブルに従って、まず最初にカーネルのオリジナルブートセクタを“0x90000”に読み込みます。
27. 次にカーネルのセットアッププログラムを“0x90200”に読み込みます。
28. 次にカーネル本体(vmlinuz)を“0x10000”に読み込みます。ただし上位アドレスに読み込むようにコンパイルされている場合は“0x100000”以降に読み込みます。
29. 最後に“0x90200”にジャンプして、カーネルのセットアッププログラムに制御を移します。

プライマリーブートローダの解析

●BIOSコールの説明

ブートローダは、ソフトウェア割り込み(INT xx)を使ってBIOSの機能を利用しています。first.Sでコールしている BIOS割り込みを次に説明します。BIOSコールでは、AHレジスタにファンクション番号を設定して割り込みを要求します(リスト1)。

●first.Sの解析

first.Sのソースコードを解析していきます。このアセンブラコードは、liloのディレクトリでmakeを実行すると、このディレクトリのMakefileに従ってアセンブルされて、バイナリコードのfirst.bが作られます。first.Sには、lilo.hがincludeされているので、defineで宣言された値が渡されます。また、liloはこのboot.bのパラメータ部分である

```
.word   VERSION
```

以降から

```
.org    CODE_START_1
```

の直前までの領域に、lilo.confを元に作成したデータを埋め込み、これをブートセクタに書き込みます。

リスト1　first.SのBIOS割り込み

```
mov  ah,#14     !#14=0x0e
int  0x10       !現在のカーソル位置に、ALレジスタの文字を表示

xor  ax,ax      !ディスクシステムのリセット(AH=0)
mov  dl,al*1    !設定 DL：ドライブ番号(ビット7が“1”ならHDD)
int  0x13       !結果 CY：“0”ならエラーなし、“1”ならエラーあり(AHにエラーコード)

mov  ah,8       !HDD情報の読み取り
int  0X13       !設定 DL：ドライブ番号：0,1,2,……(ビット7が“1”ならHDD)
                !結果 DL：連結されているドライブの数(1,2：コントローラカード0の合計)
                !        DH：ヘッド番号の最大数(0～3F)
                !        CH：シリンダ番号の最大数の低位8bit
                !        CL：セクタの最大数とシリンダ番号の最大数の高位2bit

mov  ah、0x02   !セクタの読み取り
int  0X13       !設定 AL：読み取るセクタ数(“ax=0x201”は1セクタ読み取り)
                !     DL：ドライブ番号：0,1,2,……(ビット7が“1”ならHDD)
                !     DH：ヘッド番号(0から始まり、値はチェックされない)
                !     CH：シリンダ番号の低位8bit
                !     CL：ビット7と6：シリンダ番号の高位2bit
                !          ビット5～0：セクタ番号(値はチェックされない)
                !     ES：BX バッファーアドレス
                !結果 CF：“0”ならエラーなし、“1”ならエラーあり
                !     AH：エラーコード(00H－10H、11H、12H、13H、20H、40H、80H、AAH、BBH、CCH、E0H、FFH)
```

＊1　この記述は間違いだと思う。“mov dl,#0x80”とすべき。しかし、このルーチンはあまり問題にならないだろう。

表1　liloが作成したブートセクタの構造

オフセット 16進	オフセット 10進	データ	説明
00	00	EA 69	“JMP 69”: CODE_START_1へのジャンプ命令
02	02	4C 49 4C 4F	“LILO”
06	06	01 00	STAGE_FIRST=1
08	08	14 00	VERSION =20
0A	10	5a 00	timeout
0C	12	00 00	delay
0E	14	00	Serial Port
0F	15	00	Serial Port Parameter
10	16	8B 11 69 37	time stamp
14	20	72 93 80 5B 01	Descripter Table 1st Sector Address
19	25	73 93 80 5B 01	Descripter Table 2nd Sector Address
1E	30	71 93 80 5B 01	Default Command Line Sector Address
23	35	01	Prompt Option 1=Enable
24	36	00 00	Greeting Message Length
26	38	00 00 00 00 00	Initial Greeting Message File MAP Sector Address
2b	43	75 93 80 5B 01	Key Translation Talbe Sector Address
30	48	50 93 80 5B 01	Secondary Boot Loader Sector Address 1st
35	53	51 93 80 5B 01	Secondary Boot Loader Sector Address 2nd
3A	58	52 93 80 5B 01	Secondary Boot Loader Sector Address 3rd
3F	63	53 93 80 5B 01	Secondary Boot Loader Sector Address 4th
45	68	54 93 80 5B 01	Secondary Boot Loader Sector Address 5th
4A	73	55 93 80 5B 01	Secondary Boot Loader Sector Address 6th
4E	78	56 93 80 5B 01	Secondary Boot Loader Sector Address 7th
53	83	57 93 80 5B 01	Secondary Boot Loader Sector Address 8th

表2

項目	内容	説明
ドライブ	HDD 0番	DL=0x80（ビット7が“1”なのでHDD）
ヘッド番号	0x5B	DH=0x5B
シリンダ番号	0x193	CL=0x50=0101 0000、つまりビット7＝0かつビット6＝1なので、CH=0x93、bit6*0x100+0x93=0x193
セクタ番号	0x10	CL=0x50=0101 0000で、ビット5〜ビット0がセクタ番号なので0x10

例えば、lilo.confが、

```
boot=/dev/hda3
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
image=/boot/vmlinuz-2.0.36-3vl3
     label=linux
     root=/dev/hda3
     initrd=/boot/initrd-2.0.36-3vl3.img
     read-only
other=/dev/hda1
     label=dos
     table=/dev/hda
```

となっているとすると、liloが作成したブートセクタのパラメータ部分はリスト2のようになります。

これを分かりやすく書き直したのが表1です。

LILOのVer.21では、最初のジャンプ命令以外のオフセットの値が4Bytes分大きくなります。この中で、最後の8個のセクタアドレス以外のデータは、セカンダリブートローダが使用します。セクタアドレスをBIOSに渡すには、HCS（ドライブ番号、ヘッド番号、シリンダ番号、セクタ番号、セクタ数）をDX、CX、ALの各レジスタに設定します。セクタアドレスが“50 93 80 5B 01”のとき、各レジスタには

```
CX=9350
DX=5B80
AL=1
```

の値が読み込まれます。インテル系のCPUでは、バイトの順が逆に読み込まれることに注意してください（リトルエンディアン）。

AL=1の場合は、1セクタごとに読み込まれます。lilo.confで“compact”を設定していないときは、この例のように1セクタずつ読み込みます。各レジスタの値の意味は、BIOSの説明より表2のようになります。

リスト2　liloが作成したブートセクタ（最初の部分）

```
000000   eb 69 4c 49 4c 4f 01 00 14 00 5a 00 00 00 00 00   000Cよりパラメータ
000010   8b 11 69 37 72 93 80 5b 01 73 93 80 5b 01 71 93
000020   80 5b 01 01 00 00 00 00 00 00 00 75 93 80 5b 01
000030   50 93 80 5b 01 51 93 80 5b 01 52 93 80 5b 01 53
000040   93 80 5b 01 54 93 80 5b 01 55 93 80 5b 01 56 93
000050   80 5b 01 57 93 80 5b 01 00 00 00 00 00 00 00 00
000060   00 00 00 00 00 00 00 00 00 00 00 b8 c0 07 8e d8   006bよりプログラムコード
```

表1では、セカンダリブートローダのセクタアドレスのみが、1セクタずつ増えているのが分かります。

実際のコードの説明は、first.Sのコメントとしてリスト3に掲げましたので、参考にしてください。このモジュールは、512Bytesに収めるために短くなっていることと、構造も単純になっているのとで、比較的理解しやすくなっています。

次回はsecond.Sを解説しますが、これはかなり長く、混みいっているモジュールですので、気を引き締めて頑張りましょう。

リスト3　first.Sの動作解析

```
start:  mov     ax,#BOOTSEG             !
        mov     ds,ax                   !ds=#BOOTSEG=7C0にセット
        mov     ext_es,es               !external parametersに現在のレジスタ値をセット
        mov     ext_si,si
        mov     ext_bx,bx
        mov     ext_dl,dl
        mov     ax,#FIRSTSEG
        mov     es,ax                   !es=#FIRSTSEG=9A00にセット
        mov     cx,#256                 !繰り返しの回数を256にセット
        sub     si,si                   !si=0にセット
        sub     di,di                   !di=0にセット
        cld                             !繰り返しのときに、siとdiが増加する方向にセット
        rep                             !次の命令をcx=0になるまで繰り返す
        movsw                           !ds:siが指し示すメモリから、es:diが示すメモリへ1ワードコピーし、終了後、si,diを1ワード分カウントアップ、そして、cxからは1を引く

ここまで終了すると、7c0:0～7c0:511のプログラムとデータは9A000～9A511にコピーされる

        jmpi    go,FIRSTSEG             !FIRSTSEG:goにジャンプする(次の行です)

go:     cli                             !spの変更が完了するまで割り込みを禁止する
        mov     ds,ax                   !ds=ax=FIRSTSEG=9A00
        mov     es,ax                   !es=ax=9A00
        mov     sp,#STACK               !sp=B000
        mov     ax,#STACKSEG
        mov     ss,ax                   !ss=9000
        sti                             !割り込みを許可する

        mov     al,#0x0d                !"CR(\r)"
        call    display                 !alの文字を表示するサブルーチン
        mov     al,#0x0a                !"LF(\n)"
        call    display
        mov     al,#0x4c                !"L"
        call    display                 !画面のカーソルを改行して"L"を表示

これよりセカンダリブートローダを読み込みます

lagain: mov     si,#d_addr              !#d_addr=0x30は、セカンダリブートローダの最初のセクタアドレスオフセット
        mov     bx,#SECOND              !bx=#SECOND=0x1000
        cld
sload:  lodsw                           !axにはes:siで示すメモリの内容がロードされ、siはワード分(2Bytes)プラスされる
        mov     cx,ax                   !CH:シリンダ番号の下位8bit、CL:シリンダ番号の上位2bit+セクタ番号
        lodsw                           !axには次のワードが読み込まれる
        mov     dx,ax                   !DH:ヘッド番号、DL:ドライブ番号
        or      ax,cx                   !「at EOF ?」セクタアドレスの2ワードとも"0"ならEOF、axとcxがともに"0"なら読み込み完了
        jz      done                    !「yes -> start it」前の結果が"0"ならdoneへジャンプ
        inc     si                      !セクタアドレスは5Bytes長なので1Bytesスキップし、siを次のアドレスへ移動させる
        call    cread                   !cxとdxで指定したセクタを読み込んでes:bxで指定、メモリへ書き込む
        jc      error                   !エラーのときはキャリーフラグがセットされるのでerrorへジャンプ
        add     bx,#512                 !成功したときは、書き込みメモリアドレスを1セクタ分(512Bytes)加算する
        jmp     sload                   !EOFまで繰り返す

error:                                  !エラー処理ルーチン
        xor     ax,ax                   !FDCのリセット
        mov     dl,al                   !dl=al=0(これは"mov dl,#0x80"とすべき)
        int     0x13
        jmp     lagain                  !もう一度startより繰り返す

done:   mov     al,#0x49                !"I"を表示
        call    display
        jmpi    0,SECONDSEG             !CS=9B00、IP=0に設定され、セカンダリブートローダがスタート

エラーコードを表示するサブルーチン

#ifndef LCF_NO1STDIAG
```

リスト3 （続き）

```
bout:  push    ax                  !AL=エラーコードを一時退避
       shr     al,#4               !4bit右シフトしてエラーコードの上位桁のみ取り出す
       call    nout                !最初の16進数字を表示
       pop     ax                  !エラーコードをALに復帰させて、次の数字を表示

nout:  and     al,#15              !#15=#0x0fで、下位4bitのみ有効に
       add     al,#48              !#48=#0x30を加えてASCII文字に変換
       cmp     al,#58              !#58=#0X3A(0-9か、A-Fかを判断)
       jb      nokay               !0-9ならそのまま表示
       add     al,#7               !それ以上なら7を加えて表示(0xA+0x30=0x3A、0x3A+7=-0x41、つまり"A")
nokay:
#endif
display:xor    bh,bh               !ALのASCIIコードを表示するサブルーチン
       mov     ah,#14              !#14=#0x0e
       int     0x10
       ret
```

リニアアドレスエラーのときの処理ルーチン

```
linerr:pop     dx                  !スタックを元に戻して
       pop     cx                  !call creadの後に戻る
       pop     bx
       ret                         !CY=1にセットされたまま
```

読み込みサブルーチン

```
cread: test    dl,#LINEAR_FLAG     !dl=<ドライブ番号>、ビット6が"1"でリニアアドレス
                                   !上のコードは"dl AND 0x40"を実行して結果が0だとZFがセットされる
       jz      readsect            !ZF=0、すなわちリニアアドレスではないときは、readsectへジャンプ
       and     dl,#0xff-LINEAR_FLAG !リムーブフラグ
                                   !上のコードは"dl AND 0xDF(1011 1111)"を実行してビット6を"0"にする
```

BIOSへセクタアドレスをヘッド、シリンダ、セクタ番号で渡す方法
DH：ヘッド番号(0から)
DL：ドライブ番号(0から)、ビット7が1ならHDD
シリンダ番号は10ビットなのでCHとCLに分けてセット
CH：シリンダ番号の下位8ビット
CL：シリンダ番号の上位2ビット(ビット7とビット6)、ビット5〜ビット0は6bitのセクタ番号(1〜63)

{ linear aaddress で渡す方法}
セクターアドレスを24ビットの連続番号であらわす
linear address のときはそのままlnreadへ進む
linear address ={a23,a22,a21,a20,a19,a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,a7,a6,a5,a4,a3,a2,a1,a0}
と仮定します。

リニアアドレスの変換ルーチン

```
lnread:                            !リニアアドレスの変換は、LILO Ver.21では修正されています
       push    bx                  !ES:BX=バッファアドレス
       push    cx                  !LSW：セクタアドレスの下位16ビット
                                   !DH={a23,a22,a21,a20,a19,a18,a17,a16}、DL=ドライブ番号
                                   !CH={a15,a14,a13,a12,a11,a10,a9,a8}
                                   !CL={a7,a6,a5,a4,a3,a2,a1,a0}
       push    dx                  !MSW with drive、DH=上位8ビット、DL=ドライブ番号、bxとcxとdxを一時退避
       mov     ah,#8               !ディスクの情報を得る
       int     0x13                !CXとDXにディスク情報がセットされる
       jc      linerr              !エラーのときはlinerへジャンプ

       mov     bl,dh               !BL=DH：ヘッド番号の最大数"0〜3F"
       pop     dx                  !DH={a23,a22,a21,a20,a19,a18,a17,a16}、DL=ドライブ番号
       mov     t_drive,dl          !t_drive=ドライブ番号
       mov     dl,dh               !DL=DH={a23,a22,a21,a20,a19,a18,a17,a16}
       xor     dh,dh               !DH=0
       mov     bh,dh               !BH=0、BL=ヘッド番号の最大数
       pop     ax                  !AX=CX={(a15,a14,a13,a12,11,a10,a9,a8),(a7,a6,a5,a4,a3,a2,a1,a0)}
                                   !DX:AX=24ビットのリニアアドレス
#ifdef CYL_CHECK
                                   !CX=ディスク情報={(z7,z6,z5,z4,z3,2,z1,z0),(z9,z8,  m5,m4,m3,m2,m1,m0)}
       push    cx                  !CH=シリンダ番号の最大数の下位8ビット
                                   !CL=ビット7とビット6がシリンダー番号の最大数の上位2ビット、ビット5〜ビット0がセクタの最大数
       xchg    ch,cl               !CLとCHを交換する
```

リスト3 （続き）

```
        rol     ch,1                !CH=下位8ビットのうちの、ビット7とビット6を下位に移動
        rol     ch,1                !CH={m5,m4,m3,m2,m1,m0,  z9,z8}となる
        and     ch,#3               !CH={0,0,0,0,0,0,  z9,z8}となる
        mov     n_cyl,cx            !n_cyl=CX={(0,0,0,0,0,0,  z9,z8),(z7,z6,z5,z4,z3,z2,z1,z0)}
        pop     cx                  !CX=6ステップ前でpushした値に戻す
#endif
        and     cx,#0x3f            !CH=0、CL={0,0,m5,m4,m3,m2,m1,m0 }：セクター番号の最大数
        div     cx                  !AX={(DX:AX)/CX }= リニアアドレス / 最大セクタ数
                                    !DL=割り算の剰余、DH=0 (CX <= 63)
                                    !リニアなセクタ番号をセクタ番号の最大数で割る
        inc     dl                  !セクタ番号は1から始まるので、1を加える
                                    !AX=リニアなヘッド番号(0から)
                                    !DL=セクタ番号が得られる
        mov     t_sector,dl         !t_sector=DL：セクタ番号をセーブする
        xor     dx,dx               !DX=0
        inc     bx                  !ヘッド番号は0から始まるので、ヘッド数は1を加えた値
        div     bx                  !AX=（リニアなヘッド番号 / ヘッド数）= シリンダ番号
                                    !DL= 剰余 = ヘッド番号(DH=0になる)
        mov     dh,dl               !DH= ヘッド番号セット
        mov     dl,t_drive          !DL= ドライブ番号
#ifdef CYL_CHECK
        cmp     ax,n_cyl            !セクタ数の最大数は63以下なので、AX=シリンダ番号は >= n_cyl
        ja      linerr3             !小さいときはエラーである
#endif
                                    !シリンダ番号を{0,0,0,0,0,0,c9,c8,c7,c6,c5,c4,c3,c2,c1,c0}とする
                                    !セクタ番号を   {s5,s5,s4,s3,s2,s1,s0}とする

        xchg    ah,al               !AHとALを交換：AL={0,0,0,0,0,0,c9,c8}、AH={c7,c6,c5,c4,c3,c2,c1,c0}
        ror     al,1
        ror     al,1                !AL={c9,c8,0,0,0,0,0,0}
        or      al,t_sector         !AL={c9,c8,s5,s4,s3,s2,s1,s0}
        mov     cx,ax               !CX={(c7,c6,c5,c4,c3,c2,c1,c0)、{c9,c8,s5,s4,s3,s2,s1,s0)}
        pop     bx                  !BX=バッファのオフセット
readsect:
        mov     ax,#0x201           !セクタアドレスから1セクタ読み込んで
        int     0x13                !ES:BXに書きこむ
        ret                         !成功した場合はCYフラグはリセットしてリターン
                                    !エラーの場合はCYフラグをセットしてリターン

#ifdef CYL_CHECK
linerr3:pop     bx                  !SPの位置を合わせ、サブルーチンcread内でのPUSH、POPの数を合わせる
        xor     ax,ax               !zero indicates内部エラー
        stc                         !エラー
        ret
```

R E S O U R C E

参考文献

LILO、i386アセンブラ言語、BIOS関連の参考文献を紹介しておきます。

★

●i386アセンブラ関連

[1] 80386プログラミング・ブック
B・E・Sumisu 著、荒木 訳、啓学出版

[2] 80386ハンドブック
Pen Brumm 著、古性 訳、丸善出版

●BIOS関連

[3] OADGテクニカル・リファレンス
PCオープンアーキテクチャー推進協議会

[4] The Programer's PC SOURCEBOOK
Thom Hogan著、SE編集部 訳、Microsoft Press

[5] DOS/Vプログラミングガイド
最上 著、アスキー出版

[6] PC DOS/V J6.1 BIOS技術解説書
日本IBM

[7] DOS J5.0 BIOSインターフェース技術解説書
日本IBM

●LILO関連

[8] LILO generic boot loader for Linux, Ver.20: Ueser's Guide
Wener Almesberger

[9] LILO generic boot loader for Linux, Ver.20: Technical overview
Wener almesberger

[10] LILO：the Linux Loaderの動作について
佐野武俊 著
http://www.linux.or.jp/JF/JFdocs/

[11] コメントから読むLinux カーネル
佐野武俊 著
http://www.linulx.or.jp/JF/JFdoc/readkernel.html

[12] Linux ブートローダーLILO
大崎博之 著、UNIX MAGAZINE、1998年12月号、アスキー