月例卓話

マルチブートに失敗。パソコンが動かなくなった
―「BTX-Halted」って何だろう。

黒　木　和　彦*

はじめに

PC（Personal Computer) の世界でもっと広く使われているコンピュータアーキテクチャは，インテルの IA-32 である．ここでは，その IA-32 の上にどのようにして Unix 系 OS（Operating System）のひとつである FreeBSD を乗せるのか，そのやり方のごく始めのプロセスについて書くつもりである．

技術的詳細を書く力はないので，全体の流れを書くつもりであるが，スマホにもタブレットにも SNS（Social Network Service) にも全く興味がない，いわばデジタルデバイドで格差をつけられた側の一人である私が書くのだから，不安感でいっぱいである．

なお，どの分野でもそうなのであろうが，IT の世界もまた，たくさんの略語で溢れかえっている．一種の業界用語なのだろう．これらの略語は，この分野になじみの薄い人にとっては極めて大きな参入障壁である．文句を言っても始まらないので，できるだけ正式名称を併記した．

動作モードについて

IA-32 には三つの動作モードがあるが，そのうちユーザにとって重要なのは Protected Mode と Real Address Mode である．両者の大きな違いはアドレスの計算方法にある．標準のモードは前者である．後者は初期の CPU（8086）との互換性のために用意された 20 ビットのモードであり，アドレス空間は 1 MB に限定される．現在ではシステム立ち上げ時にのみ使われている．

そして，これらのモードを使い分けなければならないことが，立ち上げに使われるプログラムをより一層複雑なものにしていると思う．古いプログラムがそのまま使えることは IA-32 の強みであると同時に，金の鎖でもあるようだ．

1. 事の始まり

現役を引退してからは，路傍の石のようなもので，取り立ててすることもなく，無聊な日々を送ることになった．

締め切りのない生活自体は大層気楽なものだが，少し工夫をしないと体も頭も早く劣化するのではないかと心配になる．少し負荷を掛けないといけない．幸いにも，体の方は，ゴルフに誘ってくれる知人がいてそれなりに動かす機会に恵まれた．残る課題は頭である．負荷になるようなものはないかと考え，思いついたのが PC の OS である．現役時代は Windows が必須であったが，路傍の石となった今は何でも自分の好きな OS を選ぶことができる．

OS に，CP/M，DOS，Windows，という流れがあるとすれば，CP/M，UNIX，LINUX，という別の流れに手を染めてみるのも悪くないと考え，UNIX 系の FreeBSD を使ってみるこ

*元三洋電機㈱研究開発本部

第 300 回京都化学者クラブ例会（平成 27 年 6 月 6 日）講演

とにした．一台の PC に二つの OS を混在させる，デュアルブートシステムの実現である．

取りあえず，自分の PC に内蔵されている 160GB の HDD（Hard Disk Drive）に空きパーティションを作り，FreeBSD をダウンロードした後，マニュアルに従ってインストールした．インストールに成功した FreeBSD は，サクサクと快適に動いたが，キーボードからコマンドを打ち込んで動かす CUI（Character-based User Interface）であり，また，アプリケーションが何も入っていないこともあって，少し使い勝手が悪い．

そこで，いろいろ調べていると，FreeBSD をベースにした SU-FreeSBIE という LIVE-CD 用のパッケージが，摂南大学工学部の井上先生の手によってアップされていることに気がついた．マウスで動かす GUI（Graphical User Interface) が使え，電子回路設計などアプリケーションソフトもひとつにまとめられており，使いやすそうなので，ダウンロードし CD に焼いた．しかし，実際に使ってみると，当然のことではあるが CD ベースのため，立ち上げに時間が掛かるのと，新たに作成したデータの保存ができない，という二つの問題に直面した．幸い，SU-FreeSBIE の中に USB メモリに落とし込むためのインストーラが予め用意されていたので，それを使って立ち上げ可能な USB メモリを作った．

これなら大丈夫直ぐに動く，と思ったのだが，予想に反して，立ち上げると，GPR（General Purpose Register) の内容をダンプした後，「BTX Halted」のメッセージを出して動きを止めてしまった．キーボードを始め一切の入力が受け付けられない．電源を切ることすらできない．最初の頃は，他にやり方を思いつかなかったので，電源ケーブルを抜いて PC を止めた．乱暴なやり方だが確実に停止できる Lethal Weapon である．一体何が起こったのか，時間はたっぷりあるし，少し調べてみようと思ったのが，事の始まりである．

ほんの少しのハードウェアの知識と，アッセンブラと，C 言語とを武器に FreeBSD のブートプロセスに挑んだわけで，まるで JABBERWOCKY を読み取ろうとした「鏡の国の Alice」のようなものだと，その無謀さに今では呆れている．

2. Bootstrap

FreeBSD を立ち上げる（Boot する）ためには，Hardware Initialization，BIOS，に始まって，Boot0，Boot1，Boot2 といくつものステップを経なければならない．最初の二つのステップ，即ち，Hardware Initialization と BIOS とには一般のユーザが手を入れることはできない．Hardware Initialization は Intel の領域であり，BIOS は BIOS 専門メーカの仕事である．従って，その内容について知る必要はないのだが，興味の趣くままに少し調べてみた．

2.1 Hardware Initialization

PC の電源スイッチを入れると，電源回路は所定の電圧が安定して出力されている事を確認した後，CPU（Central Processing Unit）に Power Good の信号を送る．

Power Good の信号を受け取った CPU は，内部に組み込まれた BIST（Built in Self Test）と呼ばれる一連のテストプロセスを実行する．BIST の詳細は不明だが，CPU 内部のマイクロコードの入力の全ての組み合わせなどもテストされるようだ．BIST をパスすると，全てのレジスタが所定の値に初期化されプログラムの実行が始まる．

CPUの動作モードを決めるCR0（Control Register 0）の初期値は60000010Hであり，Real Address Modeであることを示している．最初に実行されるアドレスはFFFFFFF0Hである．このアドレスはCPUの動作モードがReal Address Modeであるにも関わらず Protected Modeのアドレス計算方法で作られる．しかし，IA-32の一連のブートプロセスはReal Address Modeで動くため，アドレス計算方法の変更が必要である．そのためにはセグメントレジスタCSに新しい値をセットすれば良い．これは，Far Jumpという命令を実行することで可能である．

FFFFFFF0Hのアドレスは32ビットアドレス空間の最高値4GBから16バイト下がった所であり，そこには通常はFar Jump命令が置かれている．この命令により，CPUはBIOSのエントリーポイントであるPOST（Power on Self Test）へジャンプすると同時に，新しいCSがセットされるのでReal Address Modeに入ることにもなる．

2.2 BIOS-Basic Input Output System

BIOSはPCを立ち上げるための極めて優れた工夫だと思う．BIOSは，PCの立ち上げに必要なプログラムの集まりで基本的にはソフトウェアであるが，ROM（Read Only Memory)に書き込まれ予めマザーボードに組み込まれているので，ハードウェアの一部のようなものである．最近ではFlash NANDメモリを使って，アップデートを可能にしたシステムも出ているようだが，BIOSが壊れたらPCは二度と立ち上がらなくなるので，必要がない限り触れてはいけない．アップデート中に誰かが電源ケーブルを蹴飛ばしたら一巻の終わりである．

BIOSの仕事は多岐にわたる．キーボード，ディスプレイなどI/O機器の動作確認やDRAM（Dynamic Random Access Memory）のテストなどのほかに，大事な仕事として，IVT（Interrupt Vector Table）の設定がある．OSが動いていない状況では，PCに何かをさせようとすればBIOSの持つ機能を利用するしか手段がない．IVTは，BIOSの提供するLow-Level I/Oの機能の入り口のアドレス(Entry Point)を一覧表にしたようなもので，IA-32の持つINT nという割り込み命令を使ってアクセスする．

周辺機器の初期化と動作確認を終え，必要なIVTを設定したBIOSの最後の仕事は，設定したばかりのIVTの中の0x19のルーティンを呼び出し実行することである（Int 19Hのコマンドを使って割り込みをかける）．呼び出されたInt 19Hは，HDDの最初のセクタに書かれている512バイトのデータ（Boot0）をメインメモリの0x7c00に読み込んでくる．更に，読み込みが終了すれば，プログラムカウンタを0x7c00にセットして今読み込んだばかりのプログラムの先頭から実行する．どうして0x7c00に読み込むのかは定かではないが，世界中で広く使われたパソコン（PC/AT）をデザインしたIBMの技術者が決めたアドレスである．当時の標準的なメインメモリ32KB(0x8000）のトップ1KB(1024バイト：0x400バイト）を作業領域に確保したのであろう言う説は何となくうなずける．

この最初のセクタに書かれた512バイトの情報はMBR（Master Boot Record）と呼ばれ，PCを立ち上げるためには極めて重要なものである．IBMの技術者はこの重要な情報を守るため，第1トラックにはMBR以外書いてはいけないという規則も定めたようだ（第2トラック規定）．従って，第1トラックの第2セクター

以降は空である．

最近のBIOSに興味のある方にはBochsのBIOSプログラム（rombios.c）が参考になると思う．しかし，C言語で書かれた1万行にも及ぶプログラムでなかなかの難物である．

2.3 Boot0（MBR）

Int 19Hでメモリの0x7c00に読み込まれた512バイトのMBRは大きく3つの部分に分かれている．512バイトのうち，0x0から0x1bdまでの446バイトにはブートプログラムが書かれ，続く0x1beから0x1fdの64バイトはPT（Partition Table)として使われている．最後の2バイト，0x1feと0x1ffは0x55AAでなければならない．これはマジックナンバと呼ばれ，このセクタが正当なブートセクタであることを示すために使われる．PTはHDDを分割して使うための工夫で，ひとつのHDDを最大4つのパーティションに分割して使うことができる．パーティションと言っても物理的に分割するわけではなく，ソフト的に分けて使おうということだけなので，分割し直すことも容易だ．

PC/ATでは，パーティションの属性情報を記録するのに1パーティションあたり16バイトを使う．4つのパーティションだと全部で64バイトである．また，パーティションの開始位置と長さの記録には16バイトのうち各々4バイトが使われる．PC/ATが実用化された当時は，HDDの容量がさほど大きくなかったので4バイトもあれば十分だったのだが，1 TBを超える現在のHDDで使うには不十分である．そこで現在では，使用禁止になっている第1トラックの第2セクタ以降をPTに使おうという提案がなされている．この新しいPTはGPT（GUID Partition Table）と呼ばれ，新バージョンのOSでは標準でサポートされるようになってきている．

標準的なプログラムの動きで見てみると，0x7c00に読み込まれたMBRは，先ず自分自身を0x600に移し0x7c00以降を空にする．これは，このプログラムの最後で再びBIOSをコールして，0x7c00に次のステップのプログラムを読み込むためである．次に，PTをひとつひとつ順にチェックして，立ち上げ可能なパーティションを探す．それが見つかれば，マジックナンバ0x55AAが所定の位置に書かれていることを確認した後，今見つけたパーティションの先頭セクタ（Boot1）をBIOSを利用して0x7c00に読み込んでくる．標準MBRであればこれらの内容は512バイトに十分収まるのだが，FreeBSDのBoot0ではユーザプロンプトを出したりする関係で512バイトぎりぎりである．

2.4 Boot1

Boot1もまた512バイトの短いプログラムで，最後の2バイトにはマジックナンバ0x55AAが書かれている．PTはない．Boot1の仕事はBTX（Boot Extender)とBoot2とをメモリに読み込みBTXを走らせることにある．

FreeBSDのパーティションは，パーティションの先頭セクタから数えて，セクタ0にはBoot1が書かれ，セクタ1にはパーティションをアクセスするのに必要なDisk Labelと呼ばれる情報が書かれている．セクタ2からセクタ15にはBTXとBoot2とがあり，セクタ16以降はスーパーブロックと呼ばれ，ファイルシステムに関する情報が入っている．

従って，Boot1の仕事はパーティションの先頭16セクタを読み込み，BTXとBoot2とをメモリ上で再配置し，BTXを走らせれば完了するが，途中でMBRを再読み込みしたりするた

め，その動きは結構複雑である．

ロードアドレスは 0x8c00 であるが，パーティションの先頭 16 セクタには Boot1 自身も含まれるため BTX は 0x9000 にロードされることになる．BTX の先頭 0x10 バイトはヘッダなので，BTX のエントリーポイントは 0x9010 である．

2.5 BTX

BTX は，基本的には Protected Mode で動く小さな OS である．この小さな OS は Protected Mode で動くユーザプログラムと，Real Address Mode で動く BIOS とを結びつける重要な役割を担っている．BTX の提供するこの機能のおかげで，ユーザプログラムから BIOS の持つ様々な Low-LevelI/O の機能を利用することができる．BTX は Real Address Mode と Protected Mode の二つの異なる動作モードの間を往復しなければならないので，そのプログラムはかなり複雑であり，また，非常に難しいプログラミング技術が使われている．

2.6 Boot2

Boot2 は，BTX の提供する機能を利用するユーザ側のプログラム，即ち BTX のクライアントである．ブートストラップの最終段階で働く少し大きなプログラムで，Boot0 や Boot1 に比べれば相当に高度な仕事をこなす．

Boot2 は FreeBSD のファイルシステムを読み取った後，ハードディスクのディレクトリをスキャンしてローダープログラム（/boot/loader）を探し出す．そして，BIOS を利用してそれをメモリへロードした後 loader を走らせる．

loader もまた BTX のクライアントであり，システムの構成条件について書かれたファイル群を順次読み取り，それに基づいて OS の立ち上げを行う．これでブートは完了である．

なお，今回の発端となった「BTX Halted」のエラーメッセージは loader の実行時に発せられたものだと推定された．そのため SU-FreeSBIE の /boot/loader を FreeBSD8.2 のそれと置き換えることでこのトラブルを解消することができた (と思っている)．

おわりに

何もわからないまま FreeBSD の勉強を始め，一応の解決を見て，これからどうしようか，というひとつの分岐点に差し掛かっていた時なので，今までの経過をまとめる卓話の機会を頂き，感謝している．

この後どうするのか，ここで止めるのか，あるいは loader から kernel へと駒を進めるのか，後者の方に進みそうな気もする．ともあれ，合目的的に行動する事がベストだとは思わないので「これを知る者はこれを好む者に如かず，これを好む者はこれを楽しむ者に如かず」という孔子の教えを踏まえて，大いに楽しんで行きたいと思っている．