複数OSが共有するサービスのインタフェース

藤　波　順　久[†]

コンピュータシステムの基本的なサービス、例えば時刻や時間、キー入力などは、OSやその上のシステムライブラリで管理するのが普通である。しかし複数OSのユーザは、タイムゾーンの設定、キー入れ替えなどをOS毎に行わなければならず、プログラミングの際にも異なるモデルとインタフェースの理解が必要になる。本稿は、この問題を解決するインタフェースを提案する。

Interface of Services Shared by Various Operating Systems

FUJINAMI Nobuhisa[†]

Basic services of computer systems, such as time and keyboard input, are commonly managed by operating systems or system libraries on them. Then users of multiple operating systems must set up time zones, keyboard key exchange, etc. for each operating system. When programming, they must comprehend specific models and interfaces. This article proposes an interface to resolve these issues.

1. はじめに

コンピュータシステムの基本的なサービス、例えば時刻や時間、キー入力などは、OSやその上のシステムライブラリで管理するのが普通である。しかし、一台のコンピュータで複数のOSを切り替えて使っているユーザは、タイムゾーンの設定、キーボードのキー入れ替えなどをOS毎に行わなければならない。また、複数OSを扱っているプログラマは、そのOS毎に異なるモデルとインタフェースを理解してプログラミングを行う必要がある。

このような問題が起きる原因は、インタフェースをOSの上(アプリケーション寄り)に構築してしまったことにある。一種類のOSを複数のアーキテクチャのマシンで動かすという状況では、このような構成は非常に有効であった。しかし、現代のパソコンのように、(Macintoshを含めて)単一のアーキテクチャにほぼ統一されている状況では、OSの作者はOSを作るたびに同じような(アーキテクチャの欠点を隠す)プログラミングが必要になる上、OSのユーザに上記のような不便を強いる。

本発表で提案するインタフェースは、OSの下(ハードウェア寄り)に構築される。そもそも、多数のOSに共通する基本的なサービスならば、OSより下の階層に構築されるべきものである。今までそれが実現できなかったのは、OSの下の階層は、ハードウェアそのものか、ファームウェア(BIOS)の薄い層であり、ハードウェアの癖を隠せるほど充実したサービスを提供できなかったからである。

現代の仮想マシン技術を用いれば、OSの下に充実したサービスを提供し、OS(のドライバ)は単にそれをOSのインタフェースに変換するだけという方法が、現実的なものとなる。本発表では、いくつかの基本的なサービスについて、どのようなインタフェースを用意するべきか検討し、その効果について考察する。

2. 動　　機

IBM PCでOSを経由せずに時刻を知りたい場合、どのような方法があるだろうか。調べたところ☆、BIOSの時刻サービスは次のように2系統あることがわかった。

- SYSTEM TIME
  - 夜中の0時からの時間、および0時を過ぎたかどうかのフラグ
  - 時間の単位は、毎秒約18.2回(65536/1193180秒間隔)のclock tick
- REAL-TIME CLOCK
  - 年月日(年は4桁)
  - 時分秒(秒単位まで)およびサマータイムフラグ

通常は、起動時に後者を読み込んで前者を初期化し☆☆、以後は前者だけを使う(日付はフラグを見て使う

† 株式会社ソニー・コンピュータエンタテインメント
Sony Computer Entertainment Inc.

☆ 例えばRalf Brown's Interrupt List http://www.cs.cmu.edu/~ralf/files.htmlの1A00〜1A03。

☆☆ 初代IBM PCでは、後者がなかったので、DOSの起動のたびに人手で日付と時刻を入力していた。

側が管理)。clock tickが半端な間隔なので、1日の長さ(clock tickの1573040回ぶん)は23時間59分59.998秒となる。このずれは、1日の最後の1秒を約2ms短くすることで処理されている。つまり、そもそも仕様として1日の長さが変である。

最近のマシンでは、他にCPUのTime Stamp Counter[1)]、APIC Timer[1)]、マザーボードのHigh Precision Event Timer (HPET)[2)]、ACPI Power Management Timer[3)]を使い分ける。

OS上のプログラムでは、プログラマへの時間の見せ方はOS毎に似て異なる。例えば、プロセスの実行時間を得る場合、

Unix: `getrusage`
Windows NT: `GetProcessTimes`
Windows 9x: `QueryPerformanceCounter`の差で計算(プロセスの時間ではなく経過時間)
Windows 2.x〜3.1: `GetTickCount`の差で計算(49日でラップアラウンド)

のような違いがある。高級言語、例えばC言語で使う場合は、標準ライブラリの`clock`関数でラップされているので、下請け関数の名前の違いは知らなくてよい。しかし、ふるまいは違うことがある(特にWindowsの古いライブラリを使う場合)ので、意識する必要がある。

講演では述べなかったが、MS-DOS上のプログラムを使うだけで、IBM PCのclock tickの半端さを意識させられたことがある。時刻を設定した直後に時刻を読み出すと、設定した時刻以降の値が読み出されることを普通は期待する。しかし、IBM PCではそうならず、例えば10時00分に設定すると、clock tickの倍数に切り捨てられて9時59分59.981秒となってしまう。そのため、NEC PC-9800シリーズのMS-DOSで正しく動いていたプログラムが誤動作してしまった。

キーボードのキーコード(スキャンコード)についても、調べてみるといろいろな複雑さがある☆。そして、OS上でキーボードの配列を設定する方法も、一つのUnix系OS(例えばFreeBSD)でもコンソールの設定、XFree86(PC用のXサーバ)の設定、そしてX Window System上での設定の3通りがある。

このように、ハードウェアに癖があるのと同様に、OS上のサービスにも癖がある。

3. 個々のインタフェースの検討

本節では、OSの下にインタフェースを構築し、癖のないハードウェアに見せかける方法を、個々のサービスについて検討していく。

ここでは、インタフェースはI/Oポートとする。ソフトウェア割り込みを使ったハイパーバイザコールにしてもよいのだが、そうすると何かあったときに直接操作できるような穴を開けたくなる。I/Oポートにすればそのような誘惑を断つことができる。また、それならハードウェアのI/Oポートを隠蔽するのに自然であり、将来実ハードウェアを作れる可能性にも期待できる。

3.1 時刻・時間サービスの場合

時刻・時間サービスを大きく分けると、年月日、時分秒という、時刻を表すものと、ある時刻からの経過時間を表すものに分けられる。両方を持つハードウェアやOSもある。OSの下にサービスを構築する場合、どちらか一方だけで済ませることができれば、実装が簡単になる。

しかしながら、時刻・時間をきちんと管理しようとすると、一方だけではうまくいかない場合がある。例としてまず、録画予約をする場合を考えてみよう。予約したい時刻が表しているのは、普通の人が用いる年月日、時分秒であり、閏秒があったり、ユーザが時刻合わせをしたりすれば、それに連動するべきものである。

別の例として、スクリーンセーバ用のタイマを考えてみよう。設定時間が表しているものは、ユーザが最後に入力デバイスを操作してからの経過時間であり、時計が遅れているから10分進めたのに連動して短くなるべきものではない。この問題を避けるため、Unix系のadjtimeシステムコールのように、時計の進みを1%程度早めたり遅くしたりすることで、時間をかけて時刻を合わせる機能を持つOSもある。スクリーンセーバ用のタイマなら1%程度ずれても問題ないかもしれないが、例えばプログラムの実行時間を計測していたとしたら、許容できないだろう。

結局、両方のサービスが必要である。ここではそれを、カレンダ時計とシステムタイマと呼ぶ。カレンダ時計は、コンピュータの電源のon/offに関係なく、常に進み続けることが期待される。また、ユーザが時刻を設定することで時刻がずれる可能性がある。一方、システムタイマは電源を入れるたびに初期化されてよい。しかも、ユーザが値を設定することはないとしてよい。

ノートPCのように、サスペンド・レジューム機能を持つ場合はどうであろうか。カレンダ時計はもちろんサスペンド中も進むべきである。実行中のプロセスにとっては、時刻が突然進んだように見える。一方、システムタイマはサスペンドしたときの値から再開するべきである。サスペンドと似た機能として、マルチプロセスのOSでは、そのプロセスが実行されている間だけ進むタイマがあると便利だが、プロセスはOSの持つ概念であり、本サービスでは提供できない。

二つのサービスを提供することで、以下のような利点もある。

- カレンダ時計とシステムタイマで別々の精度の表現を利用できる。例えばカレンダ時計は精度を粗くする代わりに遠い過去や未来を表現できる形式

☆ 例えばhttp://www.skyfree.org/jpn/unixuser/の「スキャンコードは3度変貌する」を参照

にしてよい。

- カレンダ時計は通しの秒数ではなく、年月日、時分秒の形式で管理できるので、いつ閏秒が挿入されたか、(地方時の場合は) 夏時間の規則がいつ変更されたかを記録しておかなくても、時刻を表示できる。

閏秒について補足しておく。Unix 系の OS では普通、時刻を 1970 年 1 月 1 日 0 時からの通しの秒数で管理している。しかし、POSIX では閏秒を扱わないことになっているなど、閏秒まで実装しているものは少ない。NTP (Network Time Protocol) は閏秒を扱えるが、閏秒の間は時刻が進まない (非常にゆっくり進む) ようになっている。従ってたいていは、閏秒で挿入された 1 秒間に起きたことの時刻を正確に表現できない。現在のパソコンの時計の精度では、閏秒を時計の誤差と同じに扱っても実用上はあまり問題ないが、そもそもその時刻を表現できないのは仕様として変である。

IBM PC に特有の問題として、夏時間がある。MS-DOS および Windows では、ハードウェアクロック (IBM PC のハードウェアに内蔵されている時計をここではそう呼ぶ) を地方時に合わせることになっている。夏時間のある地方では、夏時間の始まりと終わりで、ハードウェアクロックをずらさなければならない。Windows には、これを自動的に行う機能がある (夏時間のあるタイムゾーンに設定すると、[自動的に夏時間の調整をする] というチェックボックスが現れる)。提案するサービスで動く OS と既存の Windows を共存させるためには、ハードウェアクロックを地方時に合わせる必要があるので、これに対処しなければならない。

実は、IBM PC のハードウェアクロックには、夏時間によって時刻を自動的にずらす機能があるが、以下の理由により、現在では使い物にならない。

- 組み込まれているのはアメリカ合衆国の夏時間の規則なので、他の国では異なる可能性がある。
- アメリカ合衆国の夏時間の規則は 2007 年から変わっている。

さらに悪いことに、現在ハードウェアクロックが夏時間を示しているかどうかを区別するフラグは、ハードウェアには存在しない。そのため、Windows を含む複数の OS を切り替えて使う PC では、一つの Windows でだけ [自動的に夏時間の調整をする] を有効にし (複数で有効にすると重複してずらされてしまう)、これを無効にしている Windows を含む他の OS では、時刻が 1 時間ずれる場合があるのを我慢するという使い方をよく見る。

ここまで来てわかるのは、機能が不足しているサービスの上にいくら工夫を重ねても、問題はあまり解決しないということである。間違いは、提案するサービスと既存の Windows を同列で共存させようとしたことにある。根本的に解決するには、提案するサービスの上で Windows を動かせるような仮想マシンを作ればよい。

```
0050h  QWORD  R   システム・タイマ・カウンタ読み出し
0058h  WORD   RW カレンダ時計:ミリ秒(書き込みは
0159h参照、以下同様)
005Ah  BYTE   RW カレンダ時計:分
005Bh  BYTE   RW カレンダ時計:時
005Ch  BYTE   RW カレンダ時計:日
005Dh  BYTE   RW カレンダ時計:月
005Eh  WORD   RW カレンダ時計:年
0150h  BYTE   RW システム・タイマ制御
0158h  BYTE   RW カレンダ時計制御
0159h  BYTE   R   カレンダ時計ロック
 成功すればbit7が1の値が返る カレンダ時計設定でロック開放
0159h  BYTE    W カレンダ時計設定(0058-005Fにラッチされていた値に設定)
 bit0-5がミリ秒、分、…に対応し、1なら設定、0なら捨てる
015Ah  WORD   RW カレンダ時計:時差(分単位)
015Ch  DWORD  R   カレンダ時計設定回数(排他制御用)
0250h  QWORD  RW システム・タイマ割り込み周期
0258h  WORD   RW カレンダ時計割り込み:ミリ秒
025Ah  BYTE   RW カレンダ時計割り込み:分
025Bh  BYTE   RW カレンダ時計割り込み:時
025Ch  BYTE   RW カレンダ時計割り込み:日
025Dh  BYTE   RW カレンダ時計割り込み:月
025Eh  WORD   RW カレンダ時計割り込み:年
```

図 1 PC-A801 の時刻・時間サービスの I/O ポート

それではここで例として、著者が作成中の PC-A801 エミュレータの時刻・時間サービスのインタフェースである、I/O ポートを図 1に示す。講演時には、時刻を地方時にするか協定世界時 (UTC) にするか迷っていた。地方時の便利な点は、将来的に閏年の規則が変更された場合でも、現在の日付を正しく取得できる (例えば協定世界時の 2 月 28 日 18 時が日本標準時☆の 2 月 29 日 3 時か 3 月 1 日 3 時かを計算しなくてよい) ことである。しかしながら、閏年の規則の寿命よりも、64 ビットアドレス空間の限界による PC-A801 の寿命のほうが短いかもしれないと考え直し、時刻は協定世界時とすることにした。夏時間を含めて地方時と協定世界時との時差も管理する。

時間および時刻は、使用するすべての桁を同時に読み出さないと意味がないので、I/O ポートを利用する場合は注意が必要である。例えば、年月日と時分秒を別々に読み出す場合、年月日を読み出した直後に日付が変わってしまうと、24 時間近くずれた時刻が読み出されてしまう。x86 プロセッサの I/O 命令では、32 ビットまでしか同時に読み出せないので、例えば年月日、時分秒、年月日の順に読み出して、年月日が一致していることを確認するなどの手順が必要である。

3.2 キーボードの場合

通常の使い方であれば、どのキーが押されたかと、そのときのシフト状態がわかればよさそうである。サービス側でキーリピートに対応するなら、キーが繰り返し押されたことにすればよい。通常ではない (楽しい)

☆ 法令上ほぼ同じなのは中央標準時 http://www.geocities.jp/nakanolib/rei/rm28-167.htm

使い方として、ゲームの入力がある。方向キーを押している間だけその方向に進み、離したら止まるような制御をしたいことがゲームではよくある。この場合は、対象となるキーが押されているか、離されているかを、好きなときに取得できる必要がある。

それでは、この2種類のサービスをどちらかだけで済ませることができるだろうか。実際のキーボードのハードウェアでは、一方のサービスしか提供されないのが普通であり、一方だけで済ませることは可能であるが、多少の困難を伴う。

前者でキーが押されたことのほかに離されたこともわかるようにすれば、各キーが押されているかどうかはすべてわかりそうだが、少しだけ問題がある。リセット直後の状態を知るには、そのときに押されているキーがもしあれば、短時間で順番に押された扱いにすればだいたいよいのだが、この順番と実際にキーが押された順番が異なる可能性がある。そもそもいつ押されたのかわからないのだから、入力があったことにするべきではないかもしれない。サスペンド・レジュームの場合にも似た問題がある。

一方、各キーについて押されているかどうかがわかれば、その状態変化を一定の間隔で調べることで、キーが押されたタイミングを検出することができる。しかしそのためには、タイマ割り込みなどの別のサービスを使うか、アプリケーションからポーリングする必要がある。

ここでは利便性のために両方のサービスを提供することにする。それを、キー状態変化とキー押下状態と呼ぶ。キー状態変化では、キーコードと、キーが押されたか離されたかの区別が返される。状態変化は非同期で起きるので、割り込みで状態変化を通知できるようなサービスが必要である。一方、キー押下状態の変化はまさに、キー状態変化サービスで通知されているので、キー押下状態は現在の値を返すサービスだけで十分である。

キーの入れ替えでよくあるのが、Ctrl キーと Caps キーの入れ替えのように、物理的なキーをそっくり入れ替えたいという要求である。キー状態変化とキー押下状態の両方を入れ替えることに気をつけさえすれば、サービスへの実装は容易であろう。そうではなく、シフトした状態に依存して入れ替えるような場合は、キー押下状態との整合性をとるのが難しい。特殊キーではOSに特有の処理が必要と予想されるので、これらはあきらめてOS側のサービスとして実現してもらうことにする。

キーボードには実は出力もある。シフト状態を示すインジケータである(メカニカルロックのキーボードでは不要であった)。自由度を考えて、ユーザ側でインジケータのon/offを制御できるようにする。

図2にI/Oポートの例を示す。講演ではさらに、キーリピートの動作について表1のように比較した。そもそも普通のキーにキーリピートは必要かという疑問もあり、どうあるべきかの結論はまだ出ていない。

```
0000h  BYTE   R  キーボード・マトリクス 0
  :     :     :                        :
000Fh  BYTE   R  キーボード・マトリクス 15
0100h  BYTE   R  キーボード・シリアル通信
0101h  BYTE   RW キーボード LED 制御
```

図2 PC-A801のキーボードサービスのI/Oポート

別の問題として、他の入力デバイスとの同期について触れておく。例えばマウスの操作で、シフトキーを押しながらクリックすると、通常のクリックとは異なる動作をするアプリケーションがある。もしクリックしたタイミングと、それをアプリケーションが受け取るタイミングにずれがあると、シフトキーが押されているかどうかが違ってしまうことがある(Windowsで経験した)。もしサービス側で入力のバッファリングを行うなら、同期が必要な他の入力デバイスの情報もいっしょにバッファリングする必要がある。

4. 関連技術と考察

OSの下にサービスを構築する例は、もちろん従来からある。本節ではそれらをいくつか紹介し、提案するインタフェースと比べることでその効果を考察する。

4.1 FM-8のキーボード

FM-8は、1981年に発売された富士通製の8ビットパソコンである。8ビットCPUをメインとサブの2個搭載し、画面はサブCPUが管理していた。キーボードは、サブCPUに接続された4ビットマイコンが管理していた。これには問題があり、キーが押されたことはサブCPUに通知されるが、離されたことは通知されない。ユーザがプログラムを送り込めるのはサブCPUまでで、4ビットマイコンのファームウェアは変更できない。そのため、FM-8のゲームでは、キーを離しただけでは動きが止まらず、止めるためのキーを押す必要があった。

これは、ハードウェアで厚い層を作りすぎて失敗した例と考えられる。

4.2 BIOS

BIOS (Basic Input/Output System)は、元々はCP/M☆の用語☆☆で、機種依存部分(デバイスドライバ)をまとめたものであった。

その後(1980年代以降)、NECのPC-9800シリーズ、IBM PCなどのROMに搭載されている、ハードウェアの制御プログラムもBIOSと呼ばれる。拡張ボード(ハードディスクインタフェースボードなど)を装着すると、それの制御用のBIOS ROMが追加される。OSやアプリケーションは、ソフトウェア割り込

☆ 1970年代にデジタルリサーチが開発した8ビットCPU用のOS

☆☆ チャットで「神代の時代から」というコメントがあったが、著者にはCP/Mより古い使用例を見つけられず、Wikipediaでも「The term first appeared in the CP/M operating system」となっている(http://en.wikipedia.org/wiki/BIOS)。おそらくこの用語として定着したのがCP/Mからだったのではないか。

機種	インタフェース	途中でシフトキーを押す	途中でシフトキーを離す
NEC PC-8001/8801	キー押下状態	yyyyyyyyyyyyyyyy	YYYYYYYYYYYYYYYY
NEC PC-9801	キー状態変化	yyyyyyyyYYYYYYYY	YYYYYYYYyyyyyyyy
IBM PC PS/2 キーボード	キー状態変化	yyyyyyyy	YYYYYYYYyyyyyyyy
著者の PC-A801	両方	yyyyyyyy	YYYYYYYY

表 1　キーリピートの比較

みを使ってその機能を呼び出す。IBM PC 系では、起動時の処理や起動画面、設定画面なども含めて BIOS と呼ぶようだ。

BIOS は、ハードウェアの細かい違いや、プロセッサの速さによる I/O ウェイトの数の違いを吸収する役割を果たしていた。N_{88}-BASIC(86)[☆1] も、CP/M-86 も、MS-DOS も、BIOS で機種依存部分を (ある程度) 隠していた。PC-9800 シリーズと IBM PC の BIOS は互換性がないが、SIM というソフト[☆2] で IBM PC の BIOS をシミュレートすることで、MS-DOS 汎用ではない IBM PC のアプリケーション (一部にパッチが必要) を動かすことができた。

今では、OS のインストール時・起動時にだけ使うもので、プロテクトモードで動作する OS の起動後は使われないものになってしまった。

BIOS は、OS の下にある点や目的は提案するインタフェースに似ているが、薄い層であり、その効果は限定的である。

4.3 USB レガシーキーボード

USB レガシーキーボードは、USB キーボードしかない IBM PC 系のマシンで、BIOS 設定画面や MS-DOS を使うための、BIOS とチップセットの機能である。USB キーボードからの入力を使って PS/2 キーボードをエミュレートする。PS/2 キーボードの I/O ポート (60h と 64h) をアクセスすると、Intel ICH[4] の機能により SMI (System Management Interrupt) がかかるので、BIOS の持つ SMI ハンドラで処理する。

この機能は、SMM (System Management Mode) で動くので、プロテクトモードや 64 ビットモードでも使え[☆3]、ほぼ完全なエミュレーションになっている。ソフトウェアからの見え方としては、提案するインタフェースに近い。違うのは、整理されたインタフェースにするのではなく、複雑なスキャンコードをそのままエミュレートしているところである。

USB レガシーキーボードの問題点としては、機能を有効にしていると、1 秒に 1 回くらい、0.3ms 程度の間処理が止まることが挙げられる。通常のソフトウェアからは SMM の動作を制御できないので、処理が止まっている間はどうしようもない。

4.4 EFI

EFI (Extensible Firmware Interface)[5] は、現在は Unified EFI (UEFI) として規格化されている。通常のユーザにとっては、Mac OS の起動に使われるブート・マネージャとして、ミニ OS(MSI の P35 Neo3-EFINITY というマザーボードは付属のゲーム、シェルを EFI から起動できる) として、あるいは、GUID Partition Table (GPT) という、2TB を超えるディスクのパーティション管理に使われる方法を含む規格として知られているであろう。

その実態は、Boot Services と Runtime Services を提供する Firmware Core と、3 種類の EFI Images (EFI Applications、EFI OS Loaders[☆4]、EFI Drivers) から構成される。特徴としては、プロテクトモードで動作すること、起動に必要なブロックデバイスやファイルシステムにアクセスできること、設定・起動に必要なキーボード入力、マウス入力、文字出力、グラフィクス出力に対応することが挙げられる。設定・起動の際 (OS の起動前) には自国語のメニューやメッセージが使える。イメージ読み込みに関する機能は非常に充実しており、SCSI や USB を含む各種のドライブやネットワークに対応する。

もし自国語対応を OS の上の層で行うと、OS の起動前のシステムセットアップメニュー (BIOS 設定画面) やブートセレクタでは、英語表示で我慢するか、それぞれが独自に自国語対応する必要があった。EFI はこの点では提案するインタフェースの代わりになりそうで期待できる。

EFI の弱点は、シングルプロセッサで動作し、デバイスドライバで割り込みが使えないことである。それだけでなく、OS のロード後は限られた機能しか使えないという、最大の弱点があることがわかった (表 2)。これでは代わりにならず、たいへん残念である。

4.5 仮想マシンのホスト OS

VMWare、Virtual PC などの仮想マシンソフトのユーザについて考えてみる。仮想マシンの上で動くゲスト OS から見れば、ホスト OS は OS の下の階層に当たる。そこで、キーカスタマイズ、夏時間の調整などの基本的な設定はホスト OS で行い、(複数あるかもしれない) ゲスト OS では設定をいじらないようにすれ

☆1 PC-9800 シリーズに搭載されている BASIC 言語の名前であるが、そのインタプリタは言語の実行環境であるだけでなく、OS のような役割も果たしていた。

☆2 http://www.vector.co.jp/soft/dos/util/se002710.html

☆3 USB 接続のフロッピーディスクドライブを従来のフロッピーディスクドライブに見せる機能を持つ BIOS も多いが、これはソフトウェア割り込みのレベルでのエミュレーションであり、I/O ポートを直接操作するプログラムからは認識できないし、プロテクトモードでは、リアルモードまたは仮想 8086 モードに切り替えてから BIOS を呼び出す必要がある。

☆4 EFI Applications の一種だが、EFI の Boot Services を終了して OS に制御を移す。

Name	Description
GetTime()	Returns the current time, time context, and time keeping capabilities.
SetTime()	Sets the current time and time context.
GetWakeupTime()	Returns the current wakeup alarm settings.
SetWakeupTime()	Sets the current wakeup alarm settings.
GetVariable()	Returns the value of a named variable.
GetNextVariableName()	Enumerates variable names.
SetVariable()	Sets, and if needed creates, a variable.
SetVirtualAddressMap()	Switches all runtime functions from physical to virtual addressing.
ConvertPointer()	Used to convert a pointer from physical to virtual addressing.
GetNextHighMonotonicCount()	Subsumes the platform's monotonic counter functionality.
ResetSystem()	Resets all processors and devices and reboots the system.

表2 Runtime Servicesの一覧(文献5)より)

ば、設定を各OSで行う必要があるという問題を、提案するインタフェースを使わなくても緩和できる。

OS依存のプログラムを書く頻度はそれほど高くないとすれば、実はこれだけでいいとも言える。つまり、IBM PCの複雑さをそのままエミュレートしているとか、エミュレートのためにわざわざOSを動かしているという点を除いて考えれば、OSの下にインタフェースを構築して、(ゲスト)OS非依存で行うという目的は達している。仮想マシンソフトのユーザならもうみんなやっているのかもしれない。

5. ま と め

IBM PCのハードウェアは難しい。OSを作ろうとしたり、OSなしで起動するプログラムを作ろうとしたりすると、難しさに直面する。OS上のプログラムであっても、ハードウェアの知識が必要とされることもある。誰かががんばって、単純化したハードウェアに見せかけるようにしてほしい。そして、その上でみんながOSを作るようになれば、プログラマの人生も単純になるであろう。

質疑応答(敬称略)

田中(産業技術総合研究所) このサービスはサーバでは使えない。ユーザごとに時差を変える必要がある。

著者 例えばプロセスごとに時差を変えたいのであれば、OSをそのように作ればよい。そのために必要な情報を提供するのが、提案するインタフェースの役目である。提供するのを(夏時間でない)地方時にするか協定世界時にするかは迷っている。

田中 地方時の場合は時差も取得できるようにしてほしい。

著者 もちろんそうする。

多田(電気通信大学)(座長) 結局リッチなサービスを作らなければならないのではないか。例えばNFSでマウントする場合には、マウントされるファイルシステムの機能をすべて仮想化しないと、使えない機能が出てくる。

著者 NFSでも例えば暗号化はその上で行うこともできる。提案するインタフェース上にいろいろな機能を構築できるようにしたい。

多田 チャットで、特定の時刻に起動したいという話が出ていた。

著者 それは3.1節のサービスだけでは無理で、時刻を指定して起動するサービスを用意する必要がある。

早川(拓殖大学) 仮想マシンを使うと、割り込み応答時間が予想できない。

著者 仮想マシンの仕様として定義し、保証できる応答時間を例えばI/Oポートで取得できるようにすることを考えている。

参 考 文 献

1) Intel Corporation. *Intel(R) 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, 2009. `http://www.intel.com/products/processor/manuals/index.htm` Order Number: 253668-032US

2) Intel Corporation. *IA-PC HPET (High Precision Event Timers) Specification Revision 1.0a*, 2004. `http://www.intel.com/hardwaredesign/hpetspec_1.pdf`

3) Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. *Advanced Configuration and Power Interface Specification Revision 4.0*, 2009. `http://www.acpi.info/spec.htm`

4) Intel Corporation. *Intel(R) 82801AA (ICH) and Intel(R) 82801AB (ICH0) I/O Controller Hub Datasheet*, 1999. `http://www.intel.com/design/archives/chipsets/815/index.htm` Order Number: 290655-003

5) Intel Corporation. *Extensible Firmware Interface Specification Version 1.10*, 2002. `http://www.intel.com/technology/efi/main_specification.htm`

各URLは2009年11月4日にアクセスできることを確認した。