# Linux Kernelの電力管理機構とパフォーマンス制御

三浦 広志 (Hiroshi Miura)*
miurahr@nttdata.co.jp
http://www.nttdata.co.jp/
(株)NTT データ 事業戦略部

2003年10月30日

## 概 要

Linux Kernel 2.6には、各種の省電力機構が搭載されている。新しいドライバー API、ACPI のサポート、ソフトウエアサスペンド、CPUFreq 等である。これらの機構を利用して、特に携帯型のノートパソコンに必要となる高度な電源管理を実現できる。

最近では、データセンターのコスト削減とリスク低減のためサーバの省電力化についても注目されている。カリフォルニアの電力危機を契機にして、電力管理機構やパフォーマンス制御機構はサービスプロバイダ事業者にとっても重要なテーマとなっている。

本稿では、ソフトウエアサスペンドによるハイバネーションの実現方法と、CPUFreq 機構による CPU 周波数／電圧制御の実現について紹介する。CPUFreq 機構の実装例として筆者の実装した Cyrix MediaGX/NatSemi Geode[1] ドライバおよび、Intel SpeedStep Driver を取り上げ、制御の理論と実装を説明する。

最後に、今後の Linux における電力管理とパフォーマンス制御の機能実装について課題と将来方向性について提言する。

## 1 はじめに

### 1.1 電力管理の目的

Linux Kernel における電力管理機構では、ノートパソコンの省電力というだけでなく沢山の便益を提供している。現在では、PDA や組み込みにも利用されており、多数のバッテリー駆動の機器で Linux が使われている。このような機器では、電力管理は利用時間に直結した意味合いを持っている。また PDA などでは起動時間も重要な意味合いを持っているが、サスペンド機能によって起動時間を短縮させる意味合いもある。

家電などで Linux が使われる場合でも、熱設計からの要請もあって利用しない間は電力を押さえて、利用中には適切なパフォーマンスを提供できなければならない。そのためには、デバイスの電力管理を行うことも必要となっている。

最近では、カリフォルニアの電力危機やデータセンター市場の過当競争を契機に、インターネットデータセンターにおけるラックマウントのサーバやブレードサーバの高集積化、省電力化に注目が集まっている。これはデータセンターの運営コストのかなり大きな部分をビルフロア賃貸料や電力費用が占めていることに起因している。サーバの高密度集積の阻害要因は熱であることから、まさに省電力機能によって熱を抑制できるため、省電力のための機構はサーバにおいても注目されているところである。

これらの省電力、熱量の抑制以外にも、電力管理には変わった用途もある。たとえば、ノートパソコンのようにバッテリ駆動の場合で、バッテリ残容量がわずかになった場合や異常高温時に緊急休止をしたり、サー

*(株)NTT データ 事業戦略部, 東北 Linux ユーザ連合会 運営委員, 日本 Webmin ユーザ会 監査, 横浜 Linux ユーザグループ

[1] NatSemi 社の Geode CPU を含む Information Appliance ビジネスユニットは AMD へ買収されることが 2003 年 8 月 6 日発表されている。

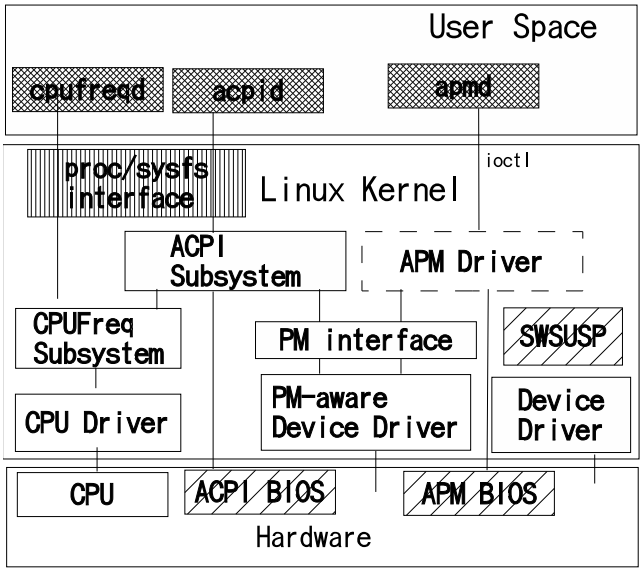


図 1: カーネルの省電力関連サブシステム

バで異常が発生したときに該当デバイスだけを停止させるなど、システム保護にも役にたたせることができる。[5]

### 1.2 システム電力管理とデバイス電力管理

電力管理にはシステム全体の管理とデバイス毎の管理の２つの種類がある。システム全体の管理は、デバイス毎の管理機構に依存している。

システム全体の管理では、サスペンド状態 (suspend-to-RAM) やハイバネーション状態 (suspend-to-disk)、電源停止状態、実行状態などのシステム全体の状態を切替える。サポート可能な状態の数はシステムアーキテクチャに依存しているが、おおよそ似通っている。

デバイス毎の管理では、CPU を実行状態や省電力状態に切り替えたり、サウンドや LAN などのデバイスについて実行中に利用されていない時に低消費電力モードに切り替えるといった制御を行なう。デバイスのサポート可能な状態の数はハードウエアやドライバに依存しているが、実行状態と停止状態はどのデバイスであっても共通にサポートされている。

### 1.3 CPUFreq 機構によるパフォーマンス制御

前述のような電力管理への要請から、コンピュータのデバイスの中でも電力消費の大きい CPU のパフォーマンスを制御することで電力消費を抑えることも行われている。特に、ニーズの大きい組み込み向け CPU やモバイル PC 向け CPU を中心に、CPU の動作パフォーマンスを動的に変更できるようなハードウエア機構が CPU LSI ベンダーによって提供されている。

例えば、モバイルパソコン向けには、Pentium III-M や Pentium-M プロセッサに採用されている Intel SpeedStep Technology、AMD 社 Athlon の PowerNow!、Transmeta 社 Crusoe の Longrun などがその代表例となっている。組み込み向けにの各種 ARM 系 CPU 等でも動作周波数の変更がサポートされている。

これらのパフォーマンス制御は、Linux Kernel 2.5/2.6 では、CPUFreq 機構と CPU 毎のドライバを使うことで制御することができる。システム負荷や電源状態（バッテリー駆動か AC 電源駆動か）にあわせた切替えには、ユーザ空間のデーモンを利用する。

## 2 Linux kernel での実装

### 2.1 ACPI の概要

ACPI とは、Advanced Configration and Power Interface の略で 1996 年に発表された電力管理とデバイス設定を行うための規格である。従来の APM(Advanced Power Management) では、BIOS により電源やパフォーマンス、デバイスが制御されているのに対して、ACPI では OS が直接電源管理を行うよう規定されている。本体装置とソフトウエアが連携して、より緻密で効果的な制御が行えるようになっている。

Linux kernel 2.6 における ACPI ドライバでは、ACPI Processor Performance States の制御を CPUFreq 機構で、ACPI System State の制御の一部 (S4 状態) の実現に SWSUSP を利用している。

### 2.2 ソフトウエアサスペンドの概要

ソフトウエアサスペンドは、APM による Suspend to Disk と同様な機能を提供する。開発当初は、APM 実

装におけるハイバネーションと Linux の相性の悪い機種用にむけたハックとして、ハイバネを提供することを目標として開発されてきた。現在、ACPI のシステム状態制御の S4 状態の実現に利用できるということで注目され、Linux Kernel 2.5/2.6 における ACPI のスリープ機構と統合して利用できるようになっている。

実際のハイバネーションを行うときには、proc インターフェースへ指示を行うことによって行われる。software suspend ネイティブのインターフェースと、ACPI sleep インターフェースの 2 つが用意されている。それぞれ、

```
# echo "0 0 2" >_
      /proc/sys/kernel/swsusp

# echo 4 > /proc/acpi/sleep
```

のように行う。実際のハイバネーションではシェルスクリプト hibernate が準備されており、サスペンドと相性の悪いネットワーク関連やリモートファイルシステムのアンマウントを、事前に行うようになっている。

## 2.3 ソフトウエアサスペンドの実装

ソフトウエアサスペンド (SWSUSP) のソースコード主要部分は、kernel/suspend.c および include/asm-i386/suspned.h include/linux/suspned.h に置かれている。現状では IDE ドライブのみのサポートであり、IA32 アーキテクチャのみをサポートしている。このアーキテクチャ依存のため、Linux のメインストリームにならない可能性は高いが、これを置き換える動作するソリューションがまだ存在していないため、有用度は高いと考える。

SWSUSP のアーキテクチャは、ユーザ空間からの指示イベントを監視する kswsusp カーネルデーモン、メモリイメージの退避を行う汎用ドライバ、アーキテクチャ依存で具体的なメモリ再配置やスワップアウトを実行するローレベルドライバで構成されている。

kswsusp カーネルデーモンから、汎用ドライバにサスペンドを指示されると、図 2 のように処理を行う。

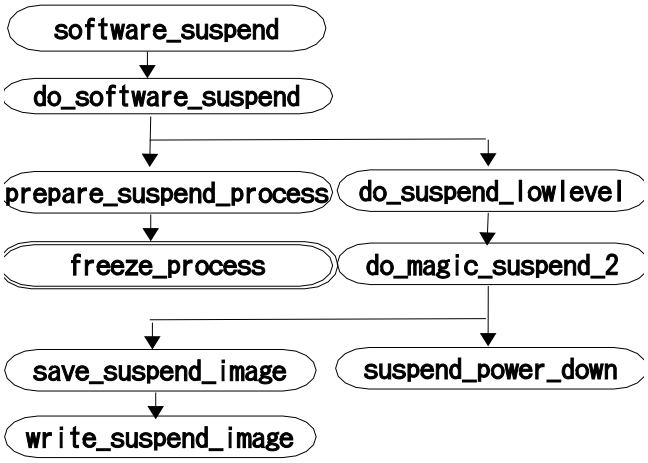


図 2: suspend 処理のプロセス

まず、software_suspend() により CPU の機能チェックを行った後、prepare_suspend_process() や freeze_process() で動いているプロセスのフリーズ処理とスワップ空間への掃き出しを行う。do_suspend_lowlevel() では、通常スワップへ掃き出されることのないページを再配置し、サスペンドイメージを作成、スワップ領域へ書き出す。最後に ACPI の S5 状態へ移行することで、電源を切断するという流れになっている。

レジューム時には、スワップパーティションに書き込まれた signature によって前回切断がハイバネーションだったか判定し、サスペンド時の逆の手順でメモリイメージを回復し、プロセスを起床させる。ユーザ空間のツールは、サスペンド以前に一旦削除したカーネルモジュールの復帰や、ネットワークインターフェースの回復、デーモンの復旧等を行って、ユーザに戻ってくると言う手順になっている。

レジューム時にほとんどのメモリは以前の状態へ回復させるが、後述する CPU パフォーマンスに関係した修正を行う必要があった。

カーネルタイマー用のグローバル定数 loops_per_jiffy についてはレジューム起動時に決定した値を利用する。これは SpeedStep など、BIOS が起動時に周波数を変更させるアーキテクチャーのサポートに必要なためだ。筆者は、CPUFreq のドライバ実装を行うにあたって、ソフトウエアサスペンドが CPU パフォーマンス制御を意識するよう修正を行った。

# 3 CPUFreq 機構

CPUFreq 機構は、Dominik Brodowski 氏[2]が開発実装を行った、Linux カーネルにおける CPU の周波数および動作電圧の動的変更のためのフレームワークである。Kernel 2.6 に正式採用されており、ACPI のパフォーマンス制御も CPUFreq 機構によって実装されている。CPUFreq では、2003 年 10 月現在、表 1 のアーキテクチャと CPU をサポートしている。

表 1: CPUFreq でサポートするアーキテクチャと CPU

| Arch | CPU |
|---|---|
| ARM | ARM Integrator |
| | ARM-SA1100 |
| | ARM-SA1110 |
| x86 | AMD Elan - SC400, SC410 |
| | AMD mobile K6-2+ |
| | AMD mobile K6-3+ |
| | AMD mobile Duron |
| | AMD mobile Athlon |
| | Cyrix Media GXm |
| | Intel mobile PIII |
| | Intel mobile PIII-M |
| | Intel Pentium 4, Intel Xeon |
| | Intel Pentium M (Centrino) |
| | NatSemi/AMD Geode GX |
| | Transmeta Crusoe |
| | VIA Cyrix 3 / C3 |
| | ACPI 2.0 互換システム |
| sparc64 | UltraSPARC-III |
| ppc | PowerBook の一部 |
| | iBook2 |
| SuperH | SH-3 |
| | SH-4 |
| x86_64 | Athlon 64 |
| | Opteron |

## 3.1 CPUFreq のユーザインターフェース

CPUFreq では、ユーザとのインターフェースは、proc インターフェースと Kernel 2.5 から導入された sysfs インターフェースの両方に対応している。推奨は sysfs インターフェースとなっており、proc インターフェースは後方互換性のために残されている。

[2]mailto:linux@brodo.de

表 2: sysfs インターフェース

| ファイル | 意味 |
|---|---|
| cpuinfo_min_freq | CPU 依存の設定可能最低周波数 |
| cpuinfo_max_freq | CPU 依存の設定可能最高周波数 |
| scaling_driver | cpufreq driver 名 |
| scaling_avalable_governors | 利用可能な governor 名 |
| scaling_governor | 現在の governor 値、動作ポリシーを意味する |
| scaling_min_freq | 現在の下限周波数 |
| scaling_max_freq | 現在の上限周波数 |
| scaling_setspeed | 現在の周波数、userspace governor 使用時の設定周波数 |

sysfs インターフェースは、cpu デバイスディレクトリの cpufreq サブディレクトリに位置している。たとえば、1 つめの CPU の場合、/sys/class/cpu/cpu0/cpufreq/ へアクセスする。ここには表 2 に示すようなインターフェースがある。

簡単には、scaling_mix_freq と scaling_max_freq へ下限、上限周波数を kHz 単位で設定し、scaling_governor へ「powersave」か「performance」を「echo」すればよい。

## 3.2 governor と Policy の制御

CPUFreq の初期の実装では、最大動作周波数、最小動作周波数をあたえるのみといった単純なものであったが、Microsoft の Windows XP に見られるようなポリシーベースの制御を行うため、Policy および Governor とよぶインターフェースを実装して利用している。

どちらを利用するかは、アーキテクチャによっている。CPU やチップセットがアグレッシブに動的にシス

テム負荷などに応じて周波数を変化させるようなアーキテクチャ（たとえば Transmeta Crusoe の LongRun）では、Policy インターフェースを使用して、システムが設定できる上限と下限をあたえる。この場合、proc および sysfs インターフェースでの Governor は効果がない。

一方、CPU やチップセットが受動的にある一つの周波数を設定するような多くのアーキテクチャでは、Governor インターフェースを利用する。proc あるいは sysfs インターフェースで紹介した「powersave」「performance」といった文字列が governor の指示である。この 2 つはカーネルにパフォーマンス設定を委ねるもので、それぞれ「最低周波数」「最高周波数」を設定させることに対応している。「userspace」を選択した場合、上限下限の間のどの周波数にするかは、ユーザやユーザ空間のデーモンによって決定されることを意味している。

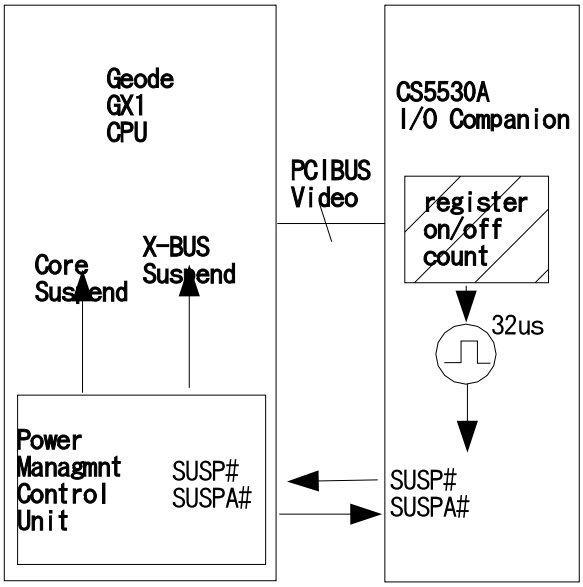


図 3: Geode CPU と Suspend Modulation

## 3.3 CPUFreq の CPU ドライバ実装の具体例

### 3.3.1 NatSemi Geode CPU

Geode GX1 CPU[4] は、旧 Cyrix 社の Media GXm CPU の後継 CPU であり、旧 Cyrix 社を買収した National Semiconductor 社によって生産されている。[3]

この CPU には、Suspend on HALT 機構とよぶ、Halt 命令を受けた時に CPU 内部のクロックを停止し、消費電力を抑える機構がある。現在、Intel Pentium III-M で利用されている Deep Sleep 状態はこの機構に酷似している。この Suspend on Halt は、halt 命令を受けた際には自動的に実行されるようになっているため、OS 側で特別な配慮をしなくても電力消費を抑えられるようになっている。この Suspend on Halt 状態にあるときには、SUSPA#ピンがアクティブになる。

CPU Susnpend 状態は、図 3 のように CPU への入力信号線の SUSP#によっても制御することができるようになっており、チップセットからの信号によって一定時間ごとに強制的に Suspend 状態にすることで、消費電力や処理性能を制御することができるようになる。これを Suspend Modulation と呼んでいる。

現在チップセットとしては、CS5530A というサウスブリッジ LSI が組み合わされ利用されている。過去には、Cyrix 社の生産した Cs5510 や Cs5520, CS5530 という LSI も使われていた。これらの LSI には、CPU の SUSP#へ接続する信号線 SUSP#があり、チップセットのレジスタに設定を行うことで、自動的に Suspend Modulation が行われる。

筆者は、この Suspend Modulation を利用した周波数制御を行う CPU ドライバを開発した。このドライバは、現行の CS5530(A) チップのほかに CS5510/5520 にも対応している。

### 3.3.2 Geode ドライバの実装

Geode CPU のドライバは、arch/i386/kernel/cpu/cpufreq/gx-suspmod.c がそのソースファイルとなっている。ドライバファイルの冒頭には、前述の Geode CPU の制御に関する理論について解説するコメントを詳細に記載しており、メンテナンスを行うプログラマがデータシートを参照しなくともドライバの動作原理を理解できるようになっている。

Geode CPU の実行周波数は、SUSP#ピンがアクティブになっている時間とインアクティブになっている時間の比によって決定される。これらの時間は、32 マイクロ秒を単位をして Suspend Moduration ON Count Reg-

[3] 2003 年 AMD 社に事業が移管される予定

ister(modon) と Suspend Moduration OFF Count Register(modoff) の 2 つの 8bit 幅のレジスタによって設定される。これらのレジスタ値から、実効的な CPU 周波数は、

$$f_{eff} = f_{gx}\frac{modon}{modon + modoff} \quad (1)$$

$$0 \leq modon, modoff \leq 255 \quad (2)$$

と表される。

実際のレジスター値の決定にあたっては、上記の計算式では自由度が高すぎるため、定数 MAXDURATION を導入して式を変形した。

$$modon + modoff \leq MAXDURATION \quad (3)$$

$$modoff = modon \times \frac{f_{gx} - f_{eff}}{f_{eff}} \quad (4)$$

さらに変形し、

$$modon = \frac{f_{eff} \times MAXDURATION}{f_{gx}} \quad (5)$$

$$modoff = MAXDURATION - modon \quad (6)$$

ここで定数 MAXDURATION は 32 マイクロ秒を単位とした Suspend 切替え周期の最大値で、値を 255 とした場合は suspend 切替え周期が 8ms になることを意味している。また、値を 255 にした場合、例えば 200MHz のプロセッサの場合、最低周波数を 781kHz に設定可能であることを意味している。ドライバでは、これをモジュールパラメータとして指定可能にしており、チューニング可能にしている。

続いてドライバ内部の実装方法を解説する。モジュールの初期化時に、cpufreq_register_driver() 関数によって cpufreq_driver 型の構造体を登録する。

リスト 1: cpufreq_driver 構造体

```
static struct cpufreq_driver
  gx_suspmod_driver = {
     .verify = cpufreq_gx_verify,
     .target = cpufreq_gx_target,
     .init   = cpufreq_gx_cpu_init,
     .name   = "gx-suspmod",
     .owner  = THIS_MODULE,
};
```

このとき、init, verify, target メンバの 3 つが重要である。init メンバへ設定する関数 cpufreq_gx_cpu_init() では、cpufreq_policy 構造体に対して、最低、最高周波数やデフォルトのポリシー、現在の動作周波数などを登録する。verify メンバへ設定する関数 cpufreq_gx_verify() では、渡された周波数が設定可能範囲に入っているかを確認し、周波数ステップにより設定可能なもっとも近い周波数への丸め込みなどを行う。設定可能周波数範囲のチェックでは、CPUFreq 機構によって準備されているライブラリ関数 cpufreq_verify_within_limits() を活用することで、簡単にチェックが可能だ。

リスト 2: 設定可能範囲チェック関数の利用

```
cpufreq_verify_within_limits(
          policy,
          (stock_freq / max_duration),
          stock_freq);
```

周波数ステップの確認では、lowlevel 関数として gx_validate_speed() 関数に選出させることで、具体的な周波数を導き出している。

target メンバへ設定する関数 cpufreq_gx_target() では、渡されたターゲット周波数への変更を実際に行っている。gx_validate_cpuspeed() 関数を活用し設定可能な周波数を導き、gx_set_cpuspeed() 関数がレジスタ書き込みを行うことで処理をおこなうようになっている。なお、gx_validate_cpuspeed() 関数は前述の数式に基づいてレジスタ値を計算する機能も持っている。

### 3.3.3 Intel モバイル Pentium III / Pentium III-M / Pentium-M の Intel SpeedStep Technology

インテル社の SpeedStep は、2003 年 8 月現在 3 つのバージョンが存在している。それぞれについて制御方法が異なっている。またチップセットによっても制御が変わってくる。[2]

初代 SpeedStep は 2000 年に発表されモバイル Pentium III の 600MHz と 650MHz 版に初めて搭載された。これは非常に複雑なシーケンスを経て周波数を切替えていることから、技術的詳細については非公開となっている。この SpeedStep は、高パフォーマンスモードと省電力モードの 2 つのモードを持っており、MS Windows ではインテルの提供するアプレットによって制御

される。切替えシーケンスの詳細は BIOS 側で実装されており、アプレットは CPU のもつ SMI/SMM インターフェースを経て BIOS の制御ルーチンを呼び出す仕組になっている。[1]

2 代目は拡張版 SpeedStep と呼ばれ、モード数には初代から変更はないが、システム負荷により動的にモードを切替えることができるようになっている。モバイル Pentium III-M CPU から搭載されるようになった。現在のところ、SpeedStep による動的切替えを Linux から利用することはできない。

最新は、インテル社の Centorino モバイルテクノロジに採用されている Intel SpeedStep Technology である。これは Pentium-M CPU と 855PM/GM チップセットの組合せによって実現されており、機能的には拡張版 SpeedStep と同等になっている。

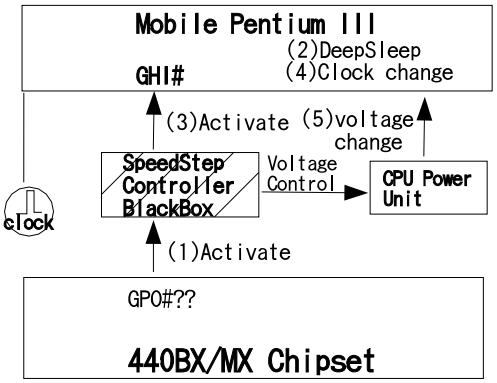


図 4: 440BX/MX に基づく SpeedStep 実装ハードウエア（想定）

### 3.3.4 speedstep-piix(-smi) ドライバの実装

従来の SpeedStep では、チップセット等によって制御方法が異なることから、それぞれ異なる CPU ドライバ (speedstep-piix4, speedstep-ich, speedstep-centorino) としている。これらのドライバでは、BIOS が準備している仕組みを利用せずに、直接チップや CPU のレジスタ設定を行うことで制御しているため、ACPI や SMI インターフェースに基づく制御と混在することができない。なお CPUFreq では、ACPI の P 状態をサポートする汎用ドライバがあり、ACPI 2.0b 仕様を実装している機器や Centrino の利用者は ACPI による制御を選択することもできる。

私は、ACPI では制御できない 440BX/MX チップセットに基づくシステムにも利用可能なドライバ (speedstep-smi) について研究開発した。

440MX/BX チップセットのドライバは、CPU 直接制御によるドライバ (speedstep-piix4) と、インテルの SpeedStep アプレットが使用している SMI インターフェースによるドライバ (speedstep-smi) が開発途上にある。前者は Dacrot Bruno 氏をリーダーとして開発しており、筆者はテストと一部コード提供を行った。SMI による SpeedStep の制御についても Kernel 2.6 に正式採用されている。現在、SMI BIOS コールに基づくドライバは各種プラットホームでの試験確認が行われ、ほぼ問題なく動作している状況となっている。

さて、440MX/BX チップセットに基づく PC の電子回路は、製造者により異なっている。そのため、前者の speedstep-piix4 ドライバでは、利用者は、リバースエンジニアリングツールを使用して接続状況を調査し、ドライバにあたえるパラーメータを決定する必要があり、通常の利用者が使うのは困難である。筆者の開発した SMI BIOS call に基づくドライバにより、440MX/BX に基づくシステムにおいても、実用的な利用が可能となった。将来、speedstep-piix4 ドライバは廃止される方向である。

440BX/MX チップセットに基づくシステムでは、図 4 のようなハードウエア構成になっていると考えられる。本チップセットの SpeedStep による周波数状態の移行は、図中の (1) のように SpeedStep のコントローラへの指示を変化させたあと、CPU を Deep Sleep 状態 (2) へと移行させることによって、電圧 (3) と周波数 (4) が変更される。

SMI インターフェースは、既定では I/O アドレス 0x00b2 番地へ out 命令を発行することにより、CPU へシステムマネジメント割り込みが発行され、システムマネジメントモード (SMM) へ移行する。SMM では、メモリアドレス体系がすべて専用のものに切替えられ、BIOS へ制御が渡されるようになっている。BIOS では、この際に渡されたレジスタ値に基づいた処理を実施することとなる。[3] 次は、ドライバにて実装した

SMI の呼び出し部分の例である。

リスト 3: SMI 呼び出し関数

```
static int ist_smi_call (
          unsigned int function,
          unsigned int state)
{
  u32         command;
  u32         new_state, result, edi;

  command = (ist_info.signature
             & 0xffffff00)
          | (smi_cmd & 0xff);

  __asm__ __volatile__("movl $-1, %%edi\n"
    "out %%al, (%%dx)\n"
    : "=a" (result), "=b" (new_state),
      "=D" (edi)
    : "a" (command), "b" (function),
      "c" (state), "d" (smi_port));
  return new_state;
}
```

# 4 結論

カーネル電源管理の概要、ソフトウエアサスペンドによる休止状態のサポートと CPUFreq 機構による CPU 周波数制御を紹介し、筆者の開発した CPUFreq ドライバについて、理論と実装を紹介した。筆者の開発したドライバや修正により、市場で多く利用されているノートパソコンで必要とされる高度な電源管理や省電力の機能を利用できる。また、Intel SpeedStep 対応機種におけるソフトウエアサスペンドにおいても、正常に利用可能になる。

今後は、ユーザランドのツールとして、電源状態、システム温度、バッテリー容量、システム負荷、利用シーンなどを総合的に判断し、最適な制御を行う仕組を研究開発していきたいと考えている。開発にあたっては、これらのパラメータによるシステム動作のモデル化とパラメータの反応係数の抽出が重要な課題になると考えている。モデルに基づく動的な係数計測と的確な制御を行うためのデーモン実装を通じて、実証を進めていきたいと考えている。

# 5 今後の課題と提言

CPUFreq, swsusp, ACPI 機構により、ノートパソコンでの利用に必要な機能はほぼ網羅された。

今後は現在実現できていない、サーバ利用のための、高度なデバイスの状態管理や CPU の状態管理、SMP におけるアンバランスな CPU のパフォーマンス制御にも挑戦したい。また、SMP における CPU の一部停止を実装することで、SMP においてもソフトウエアサスペンドが実現される可能性が高い [6] ことから、その実証実験を進めたい。

ソフトウエアサスペンド機構は、高可用性を実現するための基盤として利用することで、スムーズなノード切替えやプロセスのノード移送にも利用可能ではないかと筆者は考えている。最新のパッチにおいて、サスペンドイメージを保存するオプションが追加され、生きた状態でのマシン移送、クローンがサポートされるようになった。

最後に、本研究開発は横浜 Linux ユーザグループ[4]のカーネル読書会での議論から開始されたものであり、読書会メンバーには貴重なコメントをいただいた。また (株)NTT データ技術開発本部で行ったソフトウエアサスペンドによるノード移送の研究成果に負う部分も大きい。貴重なコメントをいただいた両グループの諸氏に謝意を表したい。本論文は、XML を使用して SmartDoc[5]プロセッサによる成形を行った。プロセッサを Free Software として提供くださっている浅海氏にも謝意を表したい。

# 参考文献

[1] Intel Corporation. *IA-32* **インテル** *(R)* **・アーキテクチャ・ソフトウエア・デベロッパーズ・マニュアル** . 2000.

[2] Intel Corporation. *Mobile Intel(R) Pentium(R) III Processor in BGA2 and Micro-PGA2 Packages at 1 GHz, 900 MHz, 850 MHz, 800 MHz, 750 MHz, 700 MHz, Low-voltage 750 MHz, Low-voltage 700 MHz, Low-voltage 600 MHz, Ultra Low-voltage 600 MHz and Ultra Low-voltage 500 MHz.* 2001.

[4] http://www.ylug.org/
[5] http://www.XMLSmartDoc.org/

[3] Microsoft Corporation. *Windows* **プラットフォーム設計メモ** *: Windows XP* **ネイティブプロセッサのパフォーマンスの制御**. 2001.

[4] National Semiconductor Corpolation. *Geode(tm)GX1 Processor Series Low Power INtegrated x86 Solution Data Sheet*. 2001.

[5] Patrick Mochel. Linux Kernel Power Management. *Proceedings of the Linux Symposium*, 2003.

[6] 菅沼公夫　河内隆仁 and 青野寛. Linux における CPU/Memory/IO の Hot Plug サポート. *Linux Conference*, 2002.