Ｐｅｎｔｉｕｍ系Ｌｉｎｕｘの詳細動作を解析する スーパートレーサとアナライザ(ＳＴＤＢ)

杉村　康　　伊土誠一

【あらまし】 オペレーティングシステム（OS）の性能の高精度な評価や動作解析には，OSの核（カーネル）の命令もトレース出来るトレーサが必要である．本論文では,Linuxのカーネルからアプリケーションプログラム（AP）迄の全てのプログラム階層に於いて全命令トレースデータの収集と解析を行うスーパートレーサとアナライザ（ STDB ; a Super Tracer and an analyzer for analyzing Detailed Behavior of a Linux on a Pentium family processor)を提案する．このスーパートレーサは，全ての割込処理，システムコール処理，並びにAP処理に於ける実行された命令，レジスタの内容等を，/dev/hdb2（IDE0 のスレーブHD）に格納する．アナライザは,それらのデータを解析して，実行されたプログラムのコール/ コールドの関係を表すネスト図等を出力する．

これらによって，8GB のIDE0-HDを増設するだけで，およそ35 MstepのOSの動作解析等が可能である．

尚，当論文の大半は,2001年4月にIEEE-ECBS2001にて既に発表を完了している．当論文は，日本の Linux関係者のために,特にIEEE Copyrights Managerより，その日本語版の再発表の許可を頂いた結果,実現したものである．

1. まえがき

近年，ソースプログラムが公開（オープンソース）され，且つ改造等が自由で無料に近い（フリーな）OSであるLinux が脚光を浴びている[1] ．フリーでオープンソースであるということは，自由に研究が可能であることも意味し，それがOSの研究者等にもたらすメリットは非常に大きい．しかし，カーネルの開発者以外の者がカーネルの動作等を解析することは，かなり困難なことであろう．何故ならば，膨大な量のプログラム中の命令の実行シーケンスを把握することは，OSの内部に精通していない者にとって，かなり困難な作業であるからだ．

一方，プロセッサやOSの性能を高精度に評価する手法として，我々は既に全命令トレース機能を持つトレーサを使用した性能評価手法[2][3][4] を提案した．Linux の急激な普及に伴って，該OS上で，上記の手法の実現が望まれている．しかしながらLinux には，それらの手法の基礎となる全命令トレース機能が存在しない．もし，その機能がLinux 上で実現出来るなら，その機能は命令の実行シーケンスを明らかに出来るので，従来の高精度な性能評価手法がLinux 上で構築可能になるだけでなく，OS自身の動作解析の容易化にも繋がるであろう．

当論文では，上記の期待に応える「Linux の詳細動作をトレースするスーパートレーサとアナライザ（STDB）」を提案する．

この全命令トレースの方式には，主に，以下の五つの方式がある．

(1) ハードウェアモニタ方式;[5],[6],[7]
(2) エミューレーションによる方式;[8]
(3) マイクロ命令方式;[9]
(4) 命令付加方式;[10],[11]
(5) シングルステップ割込方式;[2]

上記(1) は， CPUのチップの信号線より，信号を直接取込む方式である．これは最近では一次キャッシュの信号線がチップの外に出ていない為，キャッシュを停止する等のOSの変更が必要であり，又，大量のトレースデータを得る為には非常に高価なモニタを必要とするという難点がある．

上記(2) は，ソフトウェアにより，ハードウェアの動作を疑似する方法である．この手法は，カーネル部分を疑似することが非常に難しいので，全命令トレー

サには向かない.(主にAP部分のトレースにのみ使用.)

上記(3) は, CPUのマイクロ命令を変更して, 情報を収集する方式である. これは, CPU のベンダーでのみ可能であり, 当検討では実現出来ない.

上記(4) は主に縮小命令セットコンピュータ(RISC)で用いられる方式であり, 情報を取る為の命令をプログラムに埋込む方式である. この方式は, APでは命令の埋込みが自動化されているが, カーネル部分に於いては自動化された例は未だ無いので, Linux の様にカーネルのバージョンアップが頻繁に行われるOSでは, 実現が困難である. 又, 埋込まれた命令により, プログラムの大きさが2 倍以上になるので, 使用メモリ量が倍増するという欠点がある.

当検討では, Pentium family processorとRed Hat Linux を例にとって, 上記(5) 「シングルステップ割込方式」を使用し, OSの全ての部分をトレース可能とするスーパートレーサの検討と試作結果について述べる. この方式は「CPU が一つの命令を実行する直前にシングルステップ割込を発生させる」機能[12]を使用する. この方式は文献[2] に示すようにDIPS[13]というマシン上で既に実現されている. しかし該システムでは次の二つの問題点があった.

(A) 膨大な量の全命令トレースデータは, トレース中に二次記憶装置に格納する必要がある. しかし従来システム[2] では, 該情報を格納する独立発信ドライバとトレース対象のドライバとの間の競合制御を行っていない. その為以下の制約事項があった.

(a) OSが使用する二次記憶装置とトレースデータを格納する二次記憶装置を, 別のチャネル[PC-AT互換機ではInterrupt ReQuest (IRQ)]配下に置く.(コスト高になる.)

(b) 上記の二次記憶装置を, 同一チャネル配下に置くが, 二次記憶装置の割込処理はトレース対象から外し, 別途, その割込処理のみを測定して, 後日データを統合する.(工数が多くなる.)

(B) 全命令トレースを行うと, トレースされる命令の実行速度は, 全命令トレースを行わない場合に比べて, 凡そ6000分の1 以下となるが, 従来のシステムでは, 全命令トレース時にタイマーの適切な制御を実施していない. 又, 該システムでは, タイマーは主に異常処理のみでしか使用されなかった為, タイマーを停止して全命令トレースを行う場合もあった. その為タイマー処理のデータが取れなかった.

当スーパートレーサでは, 全命令トレース時に上記の制御を適切に行うロジックをカーネル内で実現することにより, 最小構成で全命令トレースが可能 [OSが使用するHDとトレースデータを格納するHDを同一 IRQ 番号配下で可能] であり, 且つタイマー処理を含んだ全命令トレースデータの収集を可能とする.

但し, 文献[14][15]に見られるような応答時間への要求が非常に厳しいシステム (ハードリアルイムシステム) では, PC-AT 互換機のシステムタイマーより高精度な処理が必要な場合がある. 当論文では, 応答時間への要求が比較的穏やかなリアルタイムシステム (ソフトリアルタイムシステム[2])やUNIXを前提としており, 当手法のハードリアルタイムシステムへの適用には, 別途, 追加検討が必要である.

尚, 全命令トレーサ全体を概観すると, 上記(1) ～(4) の方式に関しては, 上記で述べた問題点等により, OSの利用者へ殆ど普及していないと思われる. 又, 上記(5) の方式に関しては, 筆者が知る限りに於いては, ソースプログラムが非公開なOSでのみ実現されている. その為に, トレーサがOSの利用者へ開放されていないのが現状である. 何故ならば, 利用者に開放すれば, OSのノウハウが利用者に見えてしまい, 企業戦略上, それは都合が悪いからである. 当STDBは上記(5) の方式であって, 且つオープンソースのLinux 上で動作する. この場合, ソースは元々オープンであるから, OSのノウハウが利用者に見えても殆ど問題は無い. 従って, 当STDBが全世界の研究者等の間に広く普及しえる最初の全命令トレーサとなることを期待出来る. 又, そのことは, Linux の改善速度を, 更に速める結果をもたらすであろう.

一方, 世の中でトレーサと称するものには, Linux Trace Toolkit[16] やIBM のAIX[17] もある. しかし, それらは, イベントの発生だけをトレースし, 概要を把握するのに使用される. それらは, 収集情報量が非常に少ない為, 本検討が目的とするOSの動作解析や高精度な性能評価等には使用出来ない.

以下, 2.1 章でスーパートレーサに於ける全命令トレースの実現手法の概要を, 2.2 章で, 上記(A) の問題点を解決するI/O の競合防止手法を, 2.3 章で, 上

記(B) の問題点を解決するタイマー制御手法を，3 章で，全命令トレースデータを可視にするアナライザの概要を，４章で，試験環境と動作解析例について述べる．尚，以下でのLinux に関する記述は,Red Hat Linux 6.2 (英語/FTP版),カーネルは2.2.16としたものに基づいている．

２．　スーパートレーサの手法

２．１　全命令トレースの実現手法

(1) シングルステップ割込処理の割込禁止化

スーパートレーサは，トレース対象の割込処理を含む全ての命令の実行のトレースを行わなければならないので，割込禁止で走行する必要がある.(もし割込可能なら，スーパートレーサの再帰的コールが発生し，制御が困難になる.) 一方，Linux では，arch/i386/kernel/traps.cのtrap_intの中でset_trap_gate を発行することにより，シングルステップ割込処理を割込可能としている．スーパートレーサ（を導入したLinux,以下省略）では，上記をset_intr_gate に変更することにより，該割込処理を割込禁止とする．

(2) 全命令トレースデータの格納装置(/dev/hdb2) 用の独立発信ドライバ

スーパートレーサは，全命令トレースデータを/dev/hdb2 に格納するが，そのI/O 中にCPU を放棄してはならない（放棄すると，上記(1) と同様の再帰的コールの問題が発生する)． 従って，スーパートレーサは，OSのドライバとは異なるスーパートレーサ専用の独立発信ドライバを使用する． 該ドライバは，該装置へI/O を開始した場合，該装置が接続されているポートのデバイスステータスを一定間隔で読込み，ビジーが消えるのをループで待ち合わる．その後，IRQ14 の割込保留が消える迄，同ポートのデバイスステータスの読込みを繰り返し，該割込保留が消えた時点のステータスに異常が無いことを確認する． これらの手法により，CPU を放棄することなくI/O の正常完了を検知する．

尚，格納装置として/dev/hdb配下の装置を採用したのは，以下の理由による．

(A) 通常/dev/hdbは，オプションであり，大抵のPC-AT 互換機で空いていることが多い．

(B) SCSI用装置と違ってSCSIボードの増設が不要なので，低コストとなる．

一方，格納装置を,OS が使用する/dev/hdaと同じにすることは，2.2 で示す競合防止手法を使用すれば，技術的には可能である．しかし，OSと同一物理装置とすると，HDのヘッドの移動が発生する為，全命令トレースの速度が大幅に低下する．従って，別の物理装置とした．

(3) 独立発信ドライバのI/O とOSのI/O との競合防止

ある装置へのI/O の開始又は終了の処理を行っている時に別のI/O を開始すると，オーバーラン等の異常を起こす可能性がある．その為，一般に,I/Oの開始/終了処理は，CPU の状態フラグ(EFLAGS)中の割込禁止フラグをonにして行われる．スーパートレーサは，該割込禁止フラグがonの場合，全命令トレースデータをメモリ上のバッファにのみ書込み，独立発信ドライバをコールしないことにより，OSのI/O との競合を防止する．現在，このバッファは8 MBを用意している．
通常，割込禁止で走行する区間は高々１Kstep であり，１stepは平均40バイト程度の全命令トレースデータを発生する．従って，連続して200 回の割込等が起こらない限り，8 MBで十分である．

尚，現状のLinux では，物理的に連続したメモリの確保機能(kmalloc) で確保可能なメモリの最大値は，512 KBである．それを8MB に拡張する為に，以下のテーブルやルーチンを変更した．

（/usr/src/linux/以下省略）mm/page_alloc.c のNR_MEM_LISTS

mm/slab.c のSLAB_OBJ_MAX_ORDER, cache_sizes , cache_sizes_name, kmalloc , kfree

(4) シングルステップ割込の発生

以下の手法により，全命令トレース開始以降であって，CPU がアイドルでない時に走行するプログラムの実行時に，シングルステップ割込を発生させる．

(A) 全命令トレースの開始時に，全てのプロセスのタスクステートセグメント（TSS)の中及びCPU の中のEFLAGS中のトレースフラグ（TF）ビットをonとする．

(B) 全命令トレース中にEFLAGSのポップアップ命令（popf）を検出した場合は，そのEFLAGSのTFビットがoff ならonに変更する．但し，スーパートレーサのstdb_tfoffというルーチンのpopfは，全命令トレースの停止の為に存在するので，該ビットを変更しない．

(C) シングルステップ割込を除く全ての割込処理やシステムコール処理の先頭で，全命令トレース中なら，CPU のTFビットをonに変更する．但し，異常処理の例外等については，後述の(6) の(C) に従う．

(D) 全命令トレース中に割込処理からのリターン命令（iret）を検出した場合，リターン先のスタック内のTFビットがoff であって，そのときのプロセスがアイドルプロセス（プロセス名=swapper）でない場合は，該TFビットをonに変更する．

(E) 全命令トレース中にプロセスのスケジューラがアイドルプロセスでないプロセスを走行させる場合，CPU のTFビットをonとする．次に走行させるプロセスがアイドルプロセスである場合は，TFビットをoff とする．但し，アイドルプロセス走行中であっても，キャッシュや割込の後処理（ボトムハーフ処理）等を行う下記のルーチンの実行時には，プロセスをレディーにする処理等が見えるように，一時的に，TFビットをonとする．

(a) do_check_pgt_cache
(b) do_bottom_harf
(c) run_task_queue

(5) スーパートレーサのシングルステップ割込処理

シングルステップ割込発生時にはカーネルのデバグ割込ルーチンがコールされるが，その時，全命令トレース中なら，スーパートレーサのシングルステップ割込処理のルーチン（stdb_trace_do)をコールする．該ルーチンは，文献[3] と同様の性能評価を可能とし，且つ,OS の動作解析を可能とする為に，主に以下の情報を収集する．

(A) 命令アドレス
(B) 命令の内容（機械語２進データ）
(C) 変更されたレジスタ等の内容
(D) システムコール番号
(E) プロセス識別子（pid ）: マクロ名current がポイントする現プロセスのpid を1 命令毎に出力．
(F) プロセス名 :直前の命令のpid と現命令のpid が異なる時，トレースの先頭時，iret命令検出時，及びユーザ空間(<0xc0000000) の最後の命令の時に，出力．当情報は，/dev/hdb2 が満杯時で，古いデータにオーバライトすることによりトレースを継続した場合，もし，該HD上に残留する最も古い命令がユーザ空間のものであるなら，そのプロセス名を特定する為に，３章のアナライザによって，予め先読みされる．
(G) 共用ライブラリ空間(0x4??????? 及び0x5??????) とユーザプログラム空間(0x08??????)の情報（pid,プロセス名，ロードモジュール名，ロードモジュールの先頭アドレス，最終アドレス+1の情報):連続出現のユーザ空間の命令群の最初と最後の命令のみで出力.
(H) CPU を使用又は待っている（レディー）プロセス一覧: カーネル空間（>=0xc0000000）でのコール命令又はリターン命令の出現時であって，レディープロセス数が変化している場合のみ出力．
(I) 利用者が作成するown-codingデータに基づく動作解析用データ: 指定のカーネル空間のアドレスの命令が実行されようとした時に，指定のレジスタよりデータのアドレスを取込み，そのアドレス以降の指定の長さのデータを全命令トレースデータ中に出力．
（但し，データのアドレスが示すエリアがメモリ上にない場合は，データを収集しない.）

尚，上記(B) を取込む為には命令長を特定する必要がある．その為stdb_trace_do は，一命令毎に，gdb の逆アセンブラの改造版により，逆アセンブルを行う．その際に，命令のアドレスがユーザ空間であって，ページを跨がる場合等では（find_extend_vma,pgd_offset，pgd_none,pgd_bad，pmd_offset，pmd_none，pmd_bad，pte_offset の各ルーチン等を使用して）そのアドレスをカーネル空間のアドレスに変換してアクセスする．その処理に於いて，トレースされる命令が物理メモリ上にない場合は，もし，そのアドレスにstdb_trace_do がアクセスすると（該ルーチンはカーネル空間で走行するので),カーネル空間でのページフォルトが発生して，OSはpanic になってしまう．それを回避する為，該ルーチンは，命令が物理メモリ上に有るかどうかを調べ，無い場合は，一旦，何もせずにリターンする．その場合，CPU は，そのユーザ空間の命令を実行しようとするので，ユーザ空間でのページフォルトが発生し，OSのページフォルトの処理が走行し，上記のユーザ空間の命令を物理メモリ上に置く等の処理を行った後，再度上記のユーザ空間の命令が再実行されることとなる．stdb_trace_do は，それらの処理を全てトレースすることにより，上記で一旦トレースを省略したユーザ空間の命令の正常なトレースを可能に

する．

上記(G) は，３章のアナライザ用の情報であり，次の制御表を活用する．

現プロセスアドレス->mm->mmap->vm_file->f_dentry ->d_name.name， 同->mmap->vm_start，vm_end

上記(I) は，下記のスーパートレーサ用の制御表を，利用者が変更することにより行う．

arch/i386/traps.c 中のstdb_collect_data[90]= { {0,0,0 } ， {0,0,0 } ..... } ；

（注）ｎ(=0,1,2,..) ×3 番目のデータが命令のカーネルアドレス（値０はデータ終わり），ｎ×3 + 1 番目のデータは左7 ビットが取込みレジスタ（左よりeax,ebx,ecx,edx,esi,edi,ebp)，右１ビットが全命令トレースデータへの出力指示，ｎ×3 + 2 番目のデータは，出力に於けるデータ長．

(6) 全命令トレースの開始と停止等

全命令トレースの開始と停止については，従来[2] と同様のシステムコールによるもの，プログラムに変更を行う必要がない特殊キーによるもの，及びシステムの異常検出時の自動停止等を実現した．下記にそれらについて述べる．

(A) システムコールによる開始と停止

下記のシステムコールをコールすることにより，全命令トレースの開始や停止が可能である．

(a) stdb（1，全命令トレース開始通過回数，０）；

第一パラメータの１は全命令トレースの開始を意味する．第二パラメータで指定した回数分，当システムコールがコールされた時点で，全命令トレースを開始する．第三パラメータはダミーである．

(b) stdb（2，全命令トレース停止通過回数，０）；

第一パラメータの２は全命令トレースの停止を意味する．全命令トレースが開始された以降に，第二パラメータで指定した回数分，当システムコールがコールされた時点で，全命令トレースを停止する．第三パラメータはダミーである．

(c) stdb（3，全命令トレース開始通過回数，
全命令トレース停止通過回数）；

第一パラメータの３は全命令トレースの開始と停止の両者を意味する．第二パラメータで指定した回数分，当システムコールがコールされた時点で，全命令トレースを開始し，その後に，第三パラメータで指定された回数分，当システムコールがコールされた時点で，全命令トレースを停止する．

但し，利用者が必要時にのみ全命令トレースを開始することを可能とする為に，Ctrl+ Tab キーを奇数回押下した以降にのみ，上記(a) ～(c) のシステムコールが有効になる．

尚，上記の全命令トレース開始通過回数は，文献[3] に於けるウォームスタートを容易に実現する為に設けた．

上記システムコールは，現在開発中のLinux との重複を避ける為に,255というシステムコール番号を使用する．しかし,Linuxに於けるC ライブラリには登録されていない為，上記のシステムコールを記述したプログラムの先頭で，以下の記述が必要である[18]．

#include<linux/unistd.h>

_syscall3 (void, stdb, unsigned long, para1, unsigned long, para2, unsigned long, para3)；

(B) 特殊キー（Ctrl + 1）押下による開始と，特殊キー（Ctrl + Esc）押下による停止．

例えば，login コマンドのパスワード投入からプロンプト出力迄の全命令トレースを行いたい場合は，パスワードを投入してC/R を押す直前の状態で，コンソールのCtrlキーと数字の1 のキーを同時に押して，その後，C/R を投入する．又，その後のプロンプトの画面出力時に，Ctrlキーと Escキーを同時に押すことにより，全命令トレースを停止する．CPU がアイドル時には全命令トレースは行われないし，システムタイマーは凡そ6000分の１の速度で動作するので，緩慢な操作を行わない限り，上記により，ほぼ目的とする区間の全命令トレースを行うことが出来る．

(C) システムの異常を検出した時等の自動停止

全命令トレース中に，下記の例外等が発生した場合，OSのバグ等が顕在化している可能性が高いので，全命令トレースを自動停止する．( ) 内の番号は，停止時のメッセージ中に示されるendcode の値である．

(1) VM86モード検出
(2) single-step 割込以外のdebug 例外
(3) division-error例外
(4) NMI 例外
(5) int3(break point) 例外
(6) int0-overflow 例外

(7) bounds範囲外例外
(8) invalid-op例外
(9) coprocessor-segment-overrun 例外
(10) double-fault例外
(11) invalid-TSS 例外
(12) segment-not-present 例外
(13) stack-segment 例外
(14) general-protection例外
(15) spurious-interrupt例外
(16) coprocessor-error 例外
(17) alignment-check 例外
(24) トレーサのバッファオーバフロー
(29,31) panic検出

各例外の詳細については，[12]を参照されたし．

(D) /dev/hdb2 満杯時の自動停止

全命令トレースデータの格納により,/dev/hdb2が満杯になった場合，デフォルトでは，それ迄に格納したデータを残す為に，全命令トレースを自動停止する．尚，特殊キーのCtrl +^（英語キーボードではCtrl + = ）を，あらかじめ奇数回押下することにより，/dev/hdb2 が満杯になった場合に自動停止せずに，/dev/hdb2 にオーバーライトして，トレースされた命令群の最後の部分のデータを残すことも可能である．

現状では，1.4GB の全命令トレースデータを３章のアナライザで解析したときの出力ファイル容量は,6GB強に達するので，/dev/hdbとして8GB のHDを用意する場合,/dev/hdb2の大きさを，1.4GB とすることを推奨する．(1.4 + 6 + α < 8)

２．２　スーパートレーサのI/O と，同一IRQ 配下のOSのI/O との競合防止手法

(1) 何もしない場合の問題点

2.1 の(2) で述べたように，スーパートレーサは，/dev/hdb2 にアクセス中にはCPU を放棄しないので，該アクセス中にOSがI/O を開始することはない．しかしスーパートレーサがOSのI/O の開始/ 終了に全く関知しないなら，OSがI/O を開始して，それが終了する迄の間に，スーパートレーサが/dev/hdb2 にI/O を発行することが有りえる．その場合，例えばOSのI/O が/dev/hdaに対するもの（同一IRQ 配下）であるなら，スーパートレーサが/dev/hdb2 のI/O 完了をループで待合わせているときに,/dev/hda が終了割込を行おうとすることが有りえる．この場合，その終了割込を処理しない限り，スーパートレーサは,/dev/hdb2の終了を検知出来ない．しかしながら，スーパートレーサは/dev/hdaの終了割込を処理してはならない．何故ならば，/dev/hdaの割込処理は全命令トレースの対象であり，スーパートレーサが処理するとトレース抜けが発生するからである．

OSが，/dev/hdaにI/O を発行してから，その割込処理を完了する迄の間，スーパートレーサは，下記(2) の手法により，/dev/hdb2 への書込みを抑止することにより，上記の問題点を解決する．

(2) 解決手法

(a) OSは,/dev/hda へのアクセスを開始する為に，まずCPU を割込禁止にし，その後イベントを実行キューにつなぎ，I/O を開始させ，割込禁止を解除する．

(b) スーパートレーサは，それらを全てトレースし，上記の割込禁止を行った直後に，変数stdbsw_hd0ex（初期値=0）に1 を設定する．

(c) 上記の割込禁止が解除された時点で，スーパートレーサは，バッファ上に格納した全命令トレースデータを/dev/hdb2 に書込む為の処理ルーチンをコールする．その時，そのルーチンでは，stdbsw_hd0exが1である場合，ループを行って，IRQ14 が割込保留状態（/dev/hda のアクセス完了）となるのを待合わせる．その後，stdbsw_hd0exに2 を設定して，/dev/hdb2 へのアクセスを開始せずにリターンする．

(d) スーパートレーサがリターンすると，トレース対象のOSは割込可能状態であるので，OSのIRQ14 の割込処理ルーチン等が走行する．その時，スーパートレーサは，それらをトレースするが，stdbsw_hd0ex=2の場合は，メモリ上のバッファに格納するのみで，/dev/hdb2 への格納は，行わない．

(e) OSで割込処理が完了して，次のI/O の処理をする直前の命令の実行をスーパートレーサは検知し，その時点で,stdbsw_hd0ex=0 に戻して，/dev/hdb2 への格納のI/O を開始する．

尚，スーパートレーサは，全命令トレースの開始時点で，上記(a) の実行キューをチェックし，キューがある場合は，上記(b) と同等の処理を行う．

以上により，スーパートレーサのI/O と，同一IRQ 配下のOSのI/O との競合防止を実現する．

２．３　スーパートレーサのタイマー制御手法

全命令トレースを行うと，トレースされる命令の実行速度は，それを行わない場合に比べて凡そ6000分の1 以下となる．従って，何らの対策も行わないなら，相対的にタイマーの速度を6000倍に高速にした場合に等しい全命令トレースデータが収集され，該データの殆どが，タイマーの割込処理によって占められる結果となる．その弊害を完全に除去する為には，タイマーの割込頻度を6000分の1 に減速することが必要である．

一方,PC-AT互換機では,IRQ0 を使用するシステムタイマーが一般に使用される．このタイマーには,INTEL-82378[19]又は，その同等品を使用することが大半である．ところが，該タイマーに設定可能な最大の割込間隔は，凡そ55msecであり，一般にPC-AT 互換機で使用される割込頻度の5.5 分の1 迄にしか減速することが出来ない．従って，スーパートレーサに於いては，タイマーの割込周期を変更する代わりに，以下の手法によって必要な減速を実現する．

(1) カーネルでは，システムタイマー割込が発生するとシステムタイマー割込処理ルーチンへ行き，該ルーチンがタイマー処理ルーチンをコールする．

(2) スーパートレーサに於いては，システムタイマー割込が発生すると，上記の割込処理ルーチンの先頭に埋込まれているスーパートレーサのロジックを実行する．

(3) 該ロジックは，全命令トレース中であるなら，stdb_irq0_multic2 というカウンタ（初期値=0）に1 を加算し，その結果をstdb_irq0_multic0 という減速定数（現在6300）と比較する．

(4) 上記(3) の結果，カウンタの値が，減速定数以上であれば，従来のOSのシステムタイマー割込処理に制御を渡す.(その際，上記カウンタを0 に戻すし，TF ビットもonとする.)

(5) 上記(3) の結果，カウンタの値が，減速定数に達していないならば，割込コントローラ(INTEL-8259)のIRQ0のポートに割込終了(EOI)[20] を出力して，その後，OSの割込処理には制御を渡さず，割込処理からリターン(iret 命令を実行) する．

尚，減速定数はarch/i386/kernel/irq.cの中にあり，利用者は，必要に応じてその定数の変更が可能である．

３．全命令トレースデータのアナライザの概要

当アナライザは文献[2] に於ける動的ステップ(DS)自動解析手法のダウングレード版であり，以下 TCONC (Trace data CONverter's Combination)と呼ぶ．

TCONC は，次の三つの機能からなる．

- 使用論理空間解析
- 一命令毎可視データ作成とレベル解析
- ネスト図作成

以下，それぞれの概要について述べる．

３．１　使用論理空間解析

当機能は，スーパートレーサが格納した全命令トレースデータを/dev/hdb2 より入力し，プロセス毎に，実行されたユーザプログラム空間と共用ライブラリ空間に関する下記の情報を，pid_fil というファイルへ出力する．

(1) プロセス名
(2) pid
(3) ロードモジュール名
(4) 論理空間先頭アドレス
(5) 論理空間最終アドレス+1
(6) ページ切換え回数

尚，プロセス名が同一で，pid が異なる複数のプロセスが存在しえる．各論理空間は，プロセス毎に独立である．

カーネル空間に関しては，/usr/src/linux/System.map という外部名のアドレスの一覧表(mapファイル)が存在するので，TCONC はそれを利用する．しかし，ユーザ空間のプログラムのmap ファイルは存在しない．従ってユーザ空間の命令アドレスに対応する「外部名 + オフセット」を出力する為には，ユーザ空間で使用された全てのロードモジュール(LM)に対応するmap ファイルを，利用者が下記のコマンドを投入して作成し，TCONC に与える必要がある．

```
nm    LM名    > LM名.nm
sort  LM名.nm > LM名.map
```

上記のpid_fil を利用者が何も変更しない場合，次節の一命令毎可視データ作成とレベル解析は，ユーザプログラム空間のLMについては/bin配下の，共用ライブラリ空間のLMについては/lib配下の，LM名.mapのファイル名を，対応するmap ファイルと見なして入力する．それら以外の場所に格納する場合は，上記(6) の

情報を消して，その場所にパス名/ ファイル名を指定しなければならない．

共用ライブラリについては，殆どの場合/lib配下等にシンボル情報を含んだロードモジュールが存在するので，上記のnmとsortコマンドを適宜投入すればよい．

一方，ユーザプログラム空間にあるコマンド等については，残念ながら，現状では，シンボル情報を削除したロードモジュールしかOS内には存在しない．従ってコマンド等のネスト図を正確に得る必要がある場合には，先ず，シンボル情報を含んだロードモジュールを，利用者がSRPMファイルより作成する必要がある．

一方，コマンド等のネスト図を正確に得る必要がない場合（カーネル等だけが判ればよい場合）は，mapファイルとして，下記の二行からなるダミーファイルを作成すれば，上記の手間を省くことを可能とした．

論理空間先頭アドレス△? △ダミーの外部名
論理空間最終アドレス+1△A △ end
(△は１個の空白を意味する．)

この場合，pid_fil 内にある論理空間先頭と最終のアドレス+1を，利用者が正確に指定しなければならない．

３．２　一命令毎可視データ作成とレベル解析

当機能は，pid_fil，mapファイル，及び全命令トレースデータを入力し，一命令毎に逆アセンブルを行い，命令アドレスを外部名+ オフセット値に変換し，一命令毎の可視データを，ファイル名= whdb2nnn(nnn=001, 002,....）のファイルに，500Kstep毎に分割して書込む．この分割は，視認で使用するviコマンドの最大ファイル長の範囲内に，ファイル容量を抑える為である．一命令毎の可視データの例は，後述するURL のWeb サイト上のマニュアルを参照されたし．

又，次節のネスト図作成の為に，命令群を下記で述べるビッグレベルとスモールレベルに分割し，ビッグレベル毎の最小スモールレベル値を，b_lev_table のファイルに書込む．

(1) スモールレベル; 同一の外部名のルーチンで連続的に実行された命令群．

(2) ビッグレベル; 下記(A) のいずれかで始まり，下記(B) のいずれかで終わる命令群．

(A) 全命令トレースの開始時時点の先頭命令，割込処理やシステムコールからのリターン命令（iret）の次に実行された命令，割込処理で実行された最初の命令，システムコール処理で実行された最初の命令，又はプロセスの切換えの直後の命令．

(B) 全命令トレースの停止時点の命令，iret命令，又はプロセスの切換えの命令．

具体的には，各スモールレベルは，そのレベル値を持ち，

(a) ビッグレベルの開始時にそのスモールレベルにレベル値100 を与え，

(b) 他のルーチンをコールした場合，又は他のルーチンへジャンプした場合で，ジャンプ先のルーチンが，そのビッグレベルでのコールスタックにない場合は，次のスモールレベルのレベル値を現スモールレベルのレベル値+1とし，コールスタックに積む．逆に，

(c) 他のルーチンにリターン又はジャンプした場合で，リターン先がコールスタック内にある場合は，そのコールスタックのスモールレベル迄，次のスモールレベルのレベル値とコールスタックを戻す．但し，

(d) 他のルーチンにリターンした場合で，そのリターン先がコールスタックにない場合は，次のスモールレベルのレベル値を，そのビッグレベル内の最小スモールレベル値-1とし，その値で最小スモールレベル値を更新し，コールスタックには次のスモールレベルのルーチンだけを残す．

以上を各ビッグレベル毎に実施し，各ビッグレベル毎の最小スモールレベル値を求めて，上記のb_lev_table のファイルに出力し，次節のネスト図作成に渡す．

尚，共用ライブラリに於いては，

(e) リターン命令(ret) で別のルーチンを呼出す場合や，

(f)jmp命令でコールスタック内にあるルーチンを再度コールする場合がある．

それらについては，コール命令を発行するルーチン名をtconc.c 内にown-codingすることにより，レベル値を増やすことを可能とした．例として，現在，dl_runtime_resolveや init 等を，既にown-codingしている．追加が必要であれば上記の100 というレベルを超えるリターンが発生する．その場合，TCONC は，そこをイレギュラポイントとみなして，一旦ビッグレベルを分割して処理を続行する．又，下記のネスト図作成の最後で"big error= イレギュラポイント数!"のメッセージを出力し，ネスト図内に"## divided biglevel

##" の印を出力する．そのイレギュラポイント数が零でない場合，利用者は，tconc.c 内をviコマンドで，上記(e) については/userown，上記(f) については/user2own で検索の上，その箇所の例に習って own-coding を追加することにより，上記のイレギュラポイントの解消を目指すことも可能である．

３．３　ネスト図作成

当機能は，M stepのオーダーに達するトレースされた命令の流れの概要を明らかにする為に，上記のb_lev_table と全命令トレースデータを入力し，ネスト図をwnest というファイルに出力する．

ネスト図は，ビッグレベル毎の最小レベルが，行の左端になるように出力し，それよりn だけレベルが多いスモールレベルは，先頭にｎ×２の空白を挿入して出力することにより，視認性を高めた．

ネスト図の例は，後述するURL のWeb サイト上のマニュアルを参照されたし．

４．　試験環境と動作解析例

４．１　試験環境等

Red Hat Linux 6.2 英語FTP 版とSTDBを使用して，以下の三つのマシン上で，Linux コマンド（スタンドアローン），Linux ftpサーバ，Linux telnetサーバ，Linux web サーバ，Linux netscapeクライアント，及びLinux GNOME-Kterm(スタンドアローン) の環境を含む15モデルについて，延べ500Mstep以上の全命令トレースと，その結果の解析を行い，快調に動作することを確認した．

(1) 富士通 FMV-6550-DX4　　(PentiumIII 550MHz)
(2) Gateway GP-350　　(PentiumII　350MHz)
(3) Plat'Home standard　　(Pentium　200MHz)

上記に於いては，OSが使用するファイルは/dev/hda配下にあり，スーパートレーサは/dev/hdb2 を使用する．スタンドアローンでない環境に於いて使用した対向マシンは，Windows95 (Pentium 133MHz)ならびにLinux (Pentium 200MHz)であり，HUB により，トレース対象マシンと結合した．

尚，試験の大半は，Red Hat Linux 6.2 英語FTP 版で行ったが，Red Hat Linux 6.2 日本語FTP 版，Leaser5 Linux 6.2 FTP 版，Turbo Linux Workstation 日本語6.0 FTP 版，Turbo Linux Server日本語6.0 FTP 版でも，STDBに何の変更も無しに，正常に動作することを確認済である．従って，そのほかのRed Hat 系Linux でも，多分正常に動作するであろう．

４．２　動作解析例

(1) vmstatの解析例

vmstatは，どの様にしてCPU 使用率を求めているのだろうか? 以下により，その疑問を解決した．

(A) vmstat のprocps-2.0.6-5のシンボル有りRPM ファイルの作成方法が不明のため，vmstatのmap ファイルはダミーを使用．

(B)プロセスvmstatの走行状況より，160,468step ～ 3,808,808step の間に当該処理があることが判明．

(C)vmstatのユーザ空間の走行step数は2,533step のみであることがネスト図の概要情報から判明．

vmstatへの_IO_printfからのリターン箇所は以下の3 箇所．step=(a)726,642/(b)752,346/(c)3,751,173

(a) がprocs...のヘッダの出力．(b)r b w....のヘッダの出力，(c) が数値の出力と推定されるので,(b)と(c) の間で処理していると推定．

(D)上記(c) より逆方向にvmstatへのライブラリのリターン状況を調べると,step=3,664,101 で,/proc/stat をopenし,step=3,665,823 で，それをread．

(E)/proc/statのreadの中の関数のコール関係は，以下の通り．

system_call > sys_read > array_read > __get_free_pages > get_root_array > get_kstat

(F)get_kstat のソースより，CPU 使用率の情報は，kstat.cpu_user，kstat.cpu_nice，kstat.cpu_system の三つの変数より持って来る．該変数を更新している箇所は，smp.c とsched.c の二カ所．SMP ではないから，sched.c の中のupdate_process_timesというルチンが処理していると推定．そのアーギュメントticksとsystemは，外部変数lost_ticksとlost_tiks_system の値によって決まる．

(G) 上記外部変数の値を変更しているのは，do_timer．該ルーチンは，システムタイマーの割込が発生した時にコールされ，その時lost_ticksに1 を加算し，割込されたプロセスが，ユーザモードでない場合のみ，lost_ticks_system に1 を加算．

(H) それらと，上記のupdate_process_timesのロジックより，(a) システムタイマーが割込んだ時に，ユ

ーザモードでなかったら,system に1 を加算し，(b) ユーザモードでタイムスライス値が200msec 未満ならniceとsystemに1 を加算し，(c) ユーザモードでタイムスライス値が200msec 以上ならuserとsystemに1 を加算し，(d) 一定時間内のシステムタイマーの割込回数と，上記(a) ～(c) の変数の値との比率で，各 CPU 使用率を出力していることが判った.

5. むすび

上記の手法により，Pentium を使用したRed Hat Linux を例にとって，カーネルの割込処理からAP迄の全てのプログラム階層の全命令トレースを，タイマーを停止することなく，安定して，収集・解析出来ること等を示し，また，それらの使用例により，Linux のカーネル等の動作解析が可能であることを示した.

改造規模は，TCONC(新規) が8.9Kstep（うちgdb の逆アセンブラの改造版が2.8Kstep),スーパートレーサのカーネルへの新規追加ルーチン5.2Kstep（うち,gdb の逆アセンブラの改造版が2.5Kstep），既存カーネルの変更部分が0.5Kstep，合計14.6Kstep である．（うちアセンブラ記述0.3Kstep，他はC 言語記述である.）

当STDBは，Linux を開発されている方々だけでなく，Linux 上でシステムを構築されている方々や，Linux のカーネルの動作を研究したい方々にも，大いに，お役に立てるであろう.

しかしながら，STDBは，従来の機能の実現と問題点の解決等を行って，やっと基礎を築いたに過ぎない．今後の課題としては，以下が考えられる.

(1) 対称型マルチプロセッサ(SMP) のサポート.

(2) own-coding部分のコマンド化等の更なる操作性向上の実現.

(3) 収集情報の十分性の検証.

今後，上記の検討を予定している.

又，/bin配下の全てのプログラムや/lib配下の一部のプログラムは，今後，シンボル情報付のものが開放されるなら，利用者は大変助かるであろう.

尚，我々は，当STDBに関して，いくつかの特許を有している．しかしSTDBをGNU GENERAL PUBLIC LICENSE (GPL)[21][22] に基づいて，以下のWeb サイト上で公開する.

http://info.isl.ntt.co.jp/stdb/index-e.html

従ってSTDBの使用等をGPL に基づいて行う限り,NTTより特許料を請求されることは無い．又，我々は，将来STDBがLinux の一部に取入れられることを願っている．その実現の為には，Linux のコミュニティの方々による多大な御協力やリファインが必要であろう．それらが実現される迄の間，我々は，研究環境が許す限り，最新のRed Hat Linux （英語版/FTP）CD-ROMに対応するSTDBを，今後も上記URL に於いて公開を継続の予定である．何故ならば，我々は，Linux の今後の更なる発展を，心から望んでいるからである.

【 謝辞 】 以下の方々に深く感謝致します.

●シングルステップ割込機能を持つプロセッサを実現して下さった Dr. Gordon Moore 並びに彼の仲間の皆様.

●素晴らしいLinux を実現して下さったLinus Torvalds様，Linux のコミュニティの皆様，並びにGPLに基づいてプログラムを提供下さった皆様.

●fj.os.linux のnews等で情報を提供下さった皆様.

【 文 献 】

[1] John Edwards, "The Changing Face of Freeware, "Computer,vol.32,no.10,pp.11-13,October 1998.

[2] 杉村康,"マルチプロセッサのソフトオーバヘッド評価に於けるロックモデル解析の一手法,"信学論(D-I),vol.J75-D-I,no.10,pp.917-925,October 1992.

[3] Y.Sugimura,"Analysis of RISC Performance in Real-Time System,"Scripta Technica,Inc.,Systems and Computers in Japan,vol.27,no.8,pp.31-48,July 1996.（杉村康,"リアルタイムシステムに於けるRISC性能の解析,"信学論(D-I),vol.J79-D-I,no.2,pp.18-30,February 1996.）

[4] Y.Sugimura,"Avoiding Secondary Cache Conflict Misses in a Real-Time System," Scripta Technica,Inc.,Systems and Computers in Japan,vol.29,no.4,pp.71-87,April 1998.（杉村康,"リアルタイムシステムに於ける２次キャッシュ競合ミスの回避,"信学論(D-I),vol.J80-D-I,no.10,pp.813-823,October 1997.）

[5] T.Horikawa,"TOPAZ:Hardware-tracer based computer performance measurement and evaluation system," NEC Res. & Develop.,vol.33.no.4,pp.638-64

7, October 1992.
[6] 東洋テクニカ,"マルチロジックアナライザCLAS4000ユーザマニュアル,"pp.1-225,1993.
[7] Biomation Corporation,"R3000/R2000 MICROPROCESSOR ANALYSIS PACKAGE USER'S MANUAL," March 1991.
[8] Y. Sugimura,"A Verification of Accuracy in RISC Performance Analysis for Real-Time Systems," PDPTA '98 International Conference,pp.88-95, July 1998.
[9] Anant Agarwal,Richard L. Sites and Mark Horowitz,"ATUM: A New Technique for Capturing Address Traces Using Microcode,"Proceedings of the 13th Int. Symp. on Computer Arch.(ISCA-14),pp. 119-127, June 1986.
[10] T.M.Conte and C.E.Gimarc,eds."Fast Simulation of Computer Architectures,"Kluser academic Publishers,pp.5-85,Boston, 1995.
[11] J.Bradley Chen, David W.Wall and Anita Borg, "Software Methods for System Address Tracing: Implementation and Validation,"DEC WRL Research Report 95/6,http://www.research.digital.com/wrl/techreports/abstracts/89.14.html, September 1994.
[12] インテルジャパン株式会社,"Pentium ファミリーディベロッパーズマニュアル下巻: アーキテクチャとプログラミングマニュアル,"CQ出版社,p.10-4, 1995.
[13] NTT電気通信研究所,"VLSIを用いたDIPS小型システム,"通研月報,vol.36,no.10,1983.
[14] 石綿陽一,"リアルタイム処理を実現するART-Linuxの設計と実装,"CQ出版社,Interface,pp.119-129, November 1999.
[15] 竹垣盛一,"ハードリアルタイムシステムの応用例,"情報処理,vol.35,no.1,pp.34-40,1994.
[16] Karim Yaghmour,"Linux Trace Toolkit,"http://www.info.polymtl.ca/~karym/trace,2000.
[17] IBM,"AIX RISCシステム/6000 パフォーマンス監視及び調整の手引き," 1993 .
[18] R.Card/E.Dumas/F.Mevel,"Liux 2.0 カーネルブック,"オーム社,p.28,p.189,March 1999.
[19] INTEL,"Pentium and Pentium Pro Processors and Related Products,"CQ出版社,pp.2.1103-2.-1258, 1996.
[20] INTEL,"マイクロコンピュータ, ユーザーズ・マニュアル;MCS-85," インテルジャパン,p.4.77,July 1977.
[21] Free Software Foundation,Inc.,"GNU 一般公開使用承諾書,"http://www.sra.co.jp/public/doc/gnu/gpl-2j," February 2000.
[22] Free Software Foundation,Inc.,"GNU GENERAL PUBLIC LICENSE," ftp://prep.ai.mit.edu/pub/gnu/GPL, February 2000 .

【著者紹介】

杉村 康:
昭和46年鹿児島大・工学部・電子工学科卒. 同年, 日本電信電話公社（現NTT）入社. 以来ＴＳＳ系／ＴＳ系ＯＳの制御プログラム等の開発・性能評価に従事. 平成13年4月より, 国立茨城高専・電子情報工学科教授,工学博士（九大）. 電子情報通信学会, 情報処理学会, ＩＥＥＥ, 日本リヌックス協会各会員.

伊士 誠一:
昭和44年北海道大・工学部・電子工学科卒. 昭46同大大学院修士課程了. 同年日本電信電話公社（現NTT）入社. 以来, DIPS用ＯＳの研究開発, 大規模情報処理システムの開発, ソフトウェア工学, 及びデータベースの研究開発に従事. 現在ＮＴＴ情報流通基盤総合研究所長, 電子情報通信学会, 情報処理学会各会員.

[1