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）

(a Super Tracer and an analyzer for analyzing Detailed Behavior of a Linux on a Pentium family processor)

茨城工業高等専門学校・電子情報工学科 教授
(元NTT情報流通プラットフォーム研究所) 杉　村　康

NTT 情報流通基盤総合研究所　所長　伊 土　誠 一

//www.dcs.napier.ac.uk/ecbs/ecbs_2001_programe.htm

1. まえがき (1/2)

開発の背景

前提条件: **Linux** (やソフトリアルタイムシステム)
PC-AT互換機
Pentium

問題点: カーネルの全トレース機能が無!

処理能力要因を分析できない! [2,3,4]
実行ルートの特定に膨大な工数要!

カーネル～AP迄の全プログラム階層の全トレーサと, トレース結果のアナライザを作ろう!

従来方式の問題点

ハードウェアトレーサ方式 [5,6,7]	高価	×
	収集情報量小	×
エミュレーション方式 [8]	カーネルのエミュレート困難	×
マイクロコード変更方式 [9]	連続トレース数百kstep	×
	CPUベンダーのみ可能	×
コード付加方式 [10,11]	OSの膨大な変更要	×
	正常性確認工数膨大	×
	使用メモリ量が倍増	×
シングルステップ割込方式 [2]	OS使用媒体とは別IRQ要	△
	タイマー停止要	△

改善

タイマーは非トレース時と同一タイミングで動作可能 リアルタイム性有

OS使用媒体と同一IRQ配下のハードディスクに格納 最小構成で可能

2. スーパートレーサの手法(1/10)

2.1 全命令トレースの実現手法(1/8)

(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割込処理を含んだ全ての命令の実行をトレースしなければなりません．
従って，割込禁止で走行する必要があります．

しかしながら，**Linux**のシングルステップ割込処理は，**set_trap_gate**を発行することによって，割込可能状態で走行します．

スーパートレーサは，上記マクロの代わりに，**set_intr_gate**を発行することによって，シングルステップ割込処理を割込禁止で走行させます．

(2) トレースデータ格納装置(/dev/hdb2)用の独立発信ドライバ

スーパートレーサのトレースデータを格納するドライバは，I/O中に，CPUを他に渡してはなりません． さもなくば，再帰呼出し問題が発生．

スーパートレーサはOSとは独立したドライバを使用して，/dev/heb2にアクセスします．

I/Oを開始したら，定期的にデバイス状態をチェックして，デバイスビジーの終わりを待ち，その後，IRQ14の割込保留状態が消える迄，デバイス状態の読込みを行い，それが正常状態であることを確認して，I/O処理を終わります．

(3) OSのI/Oと独立発信ドライバのI/Oとの競合防止(1/2)

I/Oの開始または終了処理中に, 別のI/Oを開始するとオーバーラン等の異常発生の可能性有.

従って,一般に, I/Oの開始/終了処理中は, CPUの状態フラグ(EFLAGS)の割込禁止ビットはON.

スーパートレーサは, 該ビットがONの場合はトレースデータをメモリ上のバッファにのみ格納し, 独立発信ドライバをコールしないことにより, 別IRQ配下のOSのI/Oとの競合を防止.

現在, このバッファは8MBの連続エリア.

尚, Red Hat Linux 6.2においては, 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，512KB.

それを8MBに拡張するために，以下のテーブルやルーチンを変更.

(/usr/src/linux/) 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手法(1/2)

(A) トレース開始時に, 全プロセスのタスクステートセグメント(TSS)及びCPU中(のEFLAGS)のトレースフラグ(TF)をON.

(B) トレース中に

(a) EFLAGSのポップアップ命令(popf)検出時で, そのTFがOFFならON.

(b) 割込やシステムコール処理の先頭で, CPU中のTFをON. (異常処理用例外は6. (C) で 後述).

(c) 割込処理からのリターン命令(iret)の検出時に, リターン先スタック内のTFビットがOFFであって非アイドルプロセスなら, ONに変更.

(B) トレース中に

(d) スケジューラがアイドルプロセス以外のプロセスを走行させる場合, TFビットをON.
アイドルプロセスを走行させる場合, OFF.
但し, アイドルプロセスであっても, キャッシュや割込後処理(ボトムハーフ処理)等を行う下記のルーチン実行時には, プロセスをレディーにする(CPU待ちにする) 処理が見えるように, 一時的にTFビットをON.

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

(5) シングルステップ割込処理

カーネルデバグ割込処理ルーチンよりコールされるルーチン (stdb_trace_do)が以下の情報を収集. (詳細は, HPのマニュアル等参照.)

(A) 命令アドレスと命令内容(機械語)
(B) 変更されたレジスタの内容
(C) プロセス識別子 (pid) とプロセス名
(D) 共用ライブラリ空間(0x4..,0x5..)とユーザプログラム空間(0x8..)の情報 (ロードモジュールの名称やアドレスや長さ等)
(E) レディー状態のプロセス一覧
(F) 利用者が作成するカーネル内のOwn-cordingで指定されたアドレスのデータ.

(6) トレースの開始と停止

- (A) システムコール (stdb)
  - (a) 通過回数と開始/停止を指定.
  - (b) システムコール番号は255.
  - (c) Ctrl+Tabキーを奇数回押下時のみ有効.
- (B) 特殊キー押下
  - (a) Ctrl+1キー押下時に開始.
  - (b) Ctrl+Escキー押下時に停止.
- (C) 異常系例外やpanic発生時等に自動停止.
- (D) /dev/hdb2が満杯時に自動停止.
  但しCtrl+^キーを奇数回押下時には不停止.

この場合,トレースされた最後の部分が残る.

2.2 スーパートレーサとOSの同一IRQ配下のI/O競合防止手法

スーパートレーサはI/O時にCPUを放棄しないので, そのI/O中にOSのI/O発行は,無.

しかし, 何もしないと, OS の I/O 発行中に, スーパトレーサがI/Oを発行. この場合, OSのI/Oが先に終わると, 再帰呼出問題等が発生.

スーパートレーサは, IDE0のI/O実行イベントキューにイベントがある場合, メモリ上のバッファに書き出し, /dev/hdb2へのI/Oを開始しない.

OSのI/O割込処理の完了時に,/dev/hdb2へのI/Oを再開する.

2.3 スーパートレーサのタイマー制御手法

トレースを行うと, トレースしない場合に比べてプログラムの実行速度は, 凡そ6,000分の1.

無対策では, タイマー速度を6,000倍にした場合に等しいトレースデータが収集されてしまう.
即ち, 該データの殆どを, タイマー処理が占拠.
しかし, PC-AT互換機のタイマーの割込周期はハード仕様上, 現状の5.5倍迄にしか延長不可.

スーパートレーサは, 割込間隔を変更せずに, カウンタ(外部変数)を使用して,タイマー割込処理プログラムからタイマー処理をコールする頻度等を低下させることにより, 上記問題を解決.

3. トレースデータのアナライザの概要(1/2)

アナライザは, 文献[2]の実行ステップ自動解解析手法のマイナーバージョンであり, **TCONC (Trace data Converter‘s Conbination)**と呼ぶ.

TCONCの主な機能.

(A) トレース対象プログラムの論理空間分析

(B) トレースされた命令情報のソースファイル **(viコマンドで見れる形式)**への変換.

(C) **wnest**というファイルに, トレースされたプログラムの コール / コールド の関係等を表すネスト図等を出力.

→ 例 → 次ページ.

```
000 000001          xc010cab6 { IRQ0x05_interrupt + x1b } &&&irq5&&& : pushl
000 000002 000002 xc010ca70 { IRQ0x05_interrupt + x20 } : jmp 0xc010c910
  001 000003          xc010c910 { common_interrupt } : cld
  001 000016 000014 xc010c923 { common_interrupt + x13 } : call 0xc010ceb4
    002 000017          xc010ceb4 { do_IRQ} : pushl %ebp
    002 000029 000013 xc010ceb4 { do_IRQ + x22 } : call *%eax
      003 000030          xc010c868 { do_8259A_IRQ } : psubl $0x4,%esp
      003 000065 000036 xc010c8da { do_8259A_IRQ + x7a } : call 0xc010cfc0
        :  ( 中略)
      003 000582          xc0108869 { restore_all } : popl %ebx
      003 000592 000011 xc0108875 { restore_all + xc } : iret
```

レベル番号：
他のルーチンが呼ばれると１増える.

そのレベルの開始命令ステップ位置

そのレベルの終了命令ステップ位置

命令アドレス：
そのレベルの開始/終了命令のアドレス.

走行ステップ数：
そのレベルで走行したステップ数.

逆アセンブル結果：
その位置の命令の逆アセンブル結果.

外部名+offset：
そのレベルの開始/終了命令の外部名+offset.

図1 ネスト図の例

4. 試験環境と 動作解析例(1/3)

4.1 試験環境

- Red Hat Linux 6.2 英語FTP版とSTDBを使用.
- OSは/dev/hda, STDBは/dev/hdb2を使用.
- 下記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, 延べ500Mstep以上のトレースと, その結果解析等が快調に動作することを確認.

(1) 富士通 FMV-6550-DX4 (Pentium III 550MHz)

(2) Gateway GP-350 (Pentium II 350MHz)

(3) Plat'Home Standard (Pentium 200MHz)

4.2 動作解析例 (vmstatの解析例)(1/2)

VmstatのCPU使用率の求め方を以下により解析.

(A) ネスト図のプロセスvmstatの走行状況より, 160,468step～3,808,808stepの間にvmstatの処理があること, ネスト図の概要情報からvmstatのユーザ空間の走行step数は, 2,533stepであることが判明.

(B) _IO_printfからvmstatへのリターン箇所は, step=(a)726,642 (b) 752,346 (c) 3, 751,173の3箇所で, それぞれ, (a)が“procs…”のヘッダ出力, (b)が“r b w….”のヘッダ出力, (c) が数値の出力と思われるので, (b)と(c) の間で, CPU使用率が処理されていると推定.

(C) 上記(c) より逆方向に,ライブラリからのvmstatへ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/proc/statのファイルをread.

(D) 上記の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

4. 試験環境と 動作解析例(3/3)

4.2　動作解析例 (vmstatの解析例)(2/2)

(E) get_kstatのソースファイルより, CPU使用率の情報は, kstat.cpu_user, kstat.cpu_nice, kstat.cpu_systemの三つの変数から来る.

(F) 該変数を更新しているのは, smp.cとsched.cであったが, ここではSMPではないので, sched.cの中のupdate_process_timesで処理.

(G) 該ルーチンの引数ticksとsystemはlost_ticksとlost_ticks_systemの値によって決り, これらをdo_timerが更新.

(D) do_timerはシステムタイマー割込発生時にコールされ, その時lost_ticksに1を加算し, 割込まれたプロセスがユーザモードでない場合, lost_ticks_systemに1を加算.

以上とupdate_process_timesのロジックより,

vmstatは, 一定時間以内のシステムタイマーの割込回数に対するカーネルモード走行の比率, 及びユーザモードでタイムスライス値200msec以上の走行の比率を, CPU使用率として出力していることが判った.

5. むすび

以上により, **Red Hat Liux 6.2**を例にとって,

カーネルの割込処理～AP迄の全てのプログラム階層の解析等が, タイマーを停止すること無く, 安定して, 可能である

こと等を明らかにしました. 尚, 我々は, 当STDBに関連して幾つかの特許を有していますが, ＧＰＬ[22, 23] に基き, STDBを下記で, 公開しています.

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

よって, 当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有りません.

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LinuxTorvalds様，Linuxコミュニティーの皆様，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.

「完」