

稼動中システムのデバッグを考慮した OS デバッグ機能

畑崎 恵介[†] 中村 哲人[†] 芹沢 一^{††}

急速に利用が拡大しているオープンソースソフトウェアの代表的な OS である Linux は、24 時間 365 日稼動を要求する企業システムの運用サポートに必要なデバッグ機能に大きな課題を残している。本稿では、上記企業システムのデバッグ機能として、カーネルクラッシュダンプ機能とトレース機能を連携するデバッグ方法を提案する。さらに、トレース機能として、システムのサービスレベルの低下を最小限に抑え、かつリブートレスなトレース機能の拡張が可能な LKST (Linux Kernel State Tracer) について述べる。また、その有効性を確認するため、事例と定量的評価によって検証する。

Operating System Debugging for Running Enterprise Systems

Keisuke Hatasaki[†], Tetsuhito Nakamura[†], Kazuyoshi Serizawa^{††}

Linux is a popular open source operating system which has been used in many enterprise systems. However, it does not have enough debugging functionality to support 24x7 enterprise systems. In this paper, we propose a debugging method consisting of a cooperating kernel crash dump function and a kernel tracing function. In addition, as the tracing function, we describe LKST (Linux Kernel State Tracer), which achieves minimum tracing overhead and allows rebootless extension of trace functions. Also, we evaluate it through experiments.

1. 緒言

近年、世界の政府機関や企業システムにおいて、オープンソースの OS である Linux の利用が拡大している。Linux は、全世界のボランティア技術者が参加するコミュニティによって、その仕様決定や機能保守が進められている。しかし、企業システムでは障害発生時に迅速な対策が必須であるが、Linux には障害解析を加速するデバッグ機能については多くの課題を残している。そこで本稿では、24 時間 365 日稼動を要求する企業システムにおいて、システムの障害発生時に適切なデバッグを実現する機能に着目した。

上記企業システムのデバッグ機能には、ユーザに対するサービスレベルを保証するために (1) 性能を劣化させないこと、が重要である。さらに、システムを停止できないため、(2) システムの可用性を低下させないこと、が必要である。本稿では Linux システムにおいて、上記課題を解決する OS デバッグ機能について述べる。

2. カーネルクラッシュダンプ

2. 1. カーネルクラッシュダンプによる障害解析

一般的に利用されている運用中システムのデバッグ方法としては、カーネルクラッシュダンプがある。カーネルクラッシュダンプは、稼動中にシステムに対して影響を与えず、OS の

[†] 株式会社日立製作所 中央研究所
Hitachi Ltd. Central Research Laboratory
^{††} 株式会社日立製作所 システム開発研究所
Hitachi Ltd. System Development Laboratory

障害発生時に障害解析に必要な情報を集める手段として有効である。カーネルクラッシュダンプによるデバッグ手順は次のようになる(図1参照)。

- ①障害によりカーネルクラッシュ
- ②システムのメモリイメージを補助記憶装置(ディスク)へ保存
- ③システムの再起動
- ④保存したメモリイメージを解析

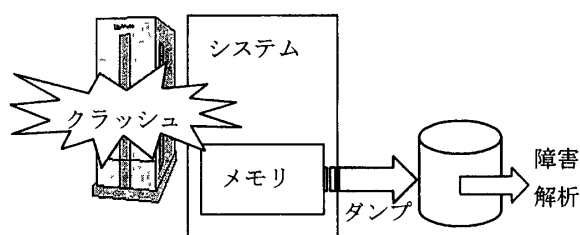


図1. カーネルクラッシュダンプ

カーネルクラッシュダンプでは、保存したメモリイメージから、障害に関係するテキストアドレス、データ、スタックの内容等を参照して、障害解析を行う。

Linux における著名なカーネルクラッシュダンプ機能には、LKCD⁵⁾(Linux Kernel Crash Dump)がある。

2. 2. カーネルクラッシュダンプの課題

カーネルクラッシュダンプは、システムが障害を発生した直後の情報だけを収集するため、障害が発生する原因となった情報が既に消滅している場合がある。例えば、次のような障害発生時には、カーネルクラッシュダンプでは原因の究明は困難である。

事例：不正メモリアドレスの参照

現象：カーネルにおいて、あるタイマをタイマリストに登録し、該タイマが起動した時点でカーネルがクラッシュ。

原因：タイマを停止することなく、タイマが登録されているタイマリストのメモリ領域が解

放されて再利用されていた。そのため、該タイマが起動されたとき、タイマの起動するハンドラへのポインタは不正な値に書き換えられているため、ページフォルトや不正命令例外が発生した(図2参照)。

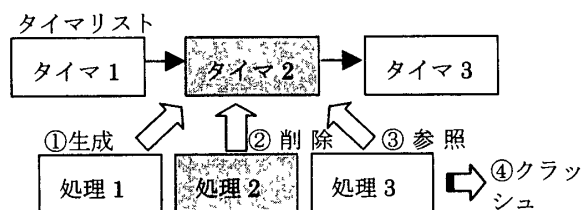


図2. ダンプで解析できない障害

この障害では、タイマリストのメモリ領域を解放した処理は、タイマが起動する前に実行されているため、障害が発生した時点のメモリイメージからは原因となった処理を検出することは困難である。そこで、カーネルクラッシュダンプに加えて、時系列的に過去に発生したカーネルの処理(イベント)を記録する機能が必要である。

3. 提案方法

3. 1. カーネルクラッシュダンプとトレース機能の連携

カーネルクラッシュダンプの欠点は、時系列的なカーネルのイベント情報を記録しておけないことである。カーネルクラッシュダンプでも、スタックトレースを利用することで、障害が発生するまでのある程度の関数をバックトレースすることが出来る。しかし、先に述べた事例では、障害の原因となる処理は、障害発生のはるか前に実行されている場合があり、スタックトレースだけでは不十分である。

そこで、カーネルの各種イベントを時系列的にメモリに記録しておく機能である、トレース機能を用意する。これにより、カーネルクラッシュダンプにより保存されたメモリイメージ

有効であることが分かる。しかし、24 時間 365 日稼動を要求するシステムにおいて、トレース機能を利用するには、下記課題を解決できなければならない。

- (1) 性能を劣化させないトレース機能の実現
- (2) システムの可用性を低下させない柔軟なデバッグ機能の実現

以下に各課題に対する対策の詳細を述べる。

4. 1. 性能対策

トレース機能は、カーネルの各種イベント処理部にフックを挿入して、イベントが発生したことをバッファに記録する。このため、トレース機能の組み込みによって、必ず性能劣化が発生する。近年のハードウェアの進歩は目覚しく、ソフトウェアの性能に関わらず高速な処理が可能になりつつあるが、トレース機能の組み込みによる性能劣化が、システムの利用者にとって無視できる程度でなければならない。そこで、3%までの性能劣化を目標にして、トレース機能の性能向上を検討した。

4. 1. 1. フックの高速化

トレース機能の性能向上を図る上で最も重要なのは、カーネルの随所に挿入されているフックの処理を高速化することである。以下ではフックの処理高速化について述べる。

■ロックレスバッファ記録

SMP マシンでは、イベントが発生したこと

をバッファへと記録する際に競合が発生しないように、ロックを取得する必要がある。しかし、ロックの処理は必ずキャッシュを超えてメモリチップへのアクセスが発生するため、性能への影響が著しい。そこで、バッファを CPU 個分用意して、各 CPU のイベントは専用のバッファへと記録することで、ロックを利用せずにバッファへと記録する方式を採用している (図 6 参照)。

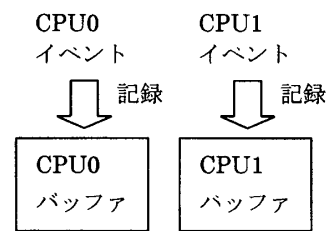


図 6. ロックレスバッファ記録

■フック関数呼び出しの高速化

LKST では、イベントマスクテーブルによって、バッファに記録するイベントを選択可能である (詳細は後述)。ここで、フックにおいてフック関数を呼び出すか否かを決定するため、メモリ上のイベントマスクテーブルを毎回参照しては、大きな性能劣化が発生する。そこで、フックの呼び出し部分に、オープンソースのプロジェクトである **Kernel Hooks** を利用した。

図 7 はカーネルのフック関数呼び出し部分のアセンブラを示したものである。図 7 に示す

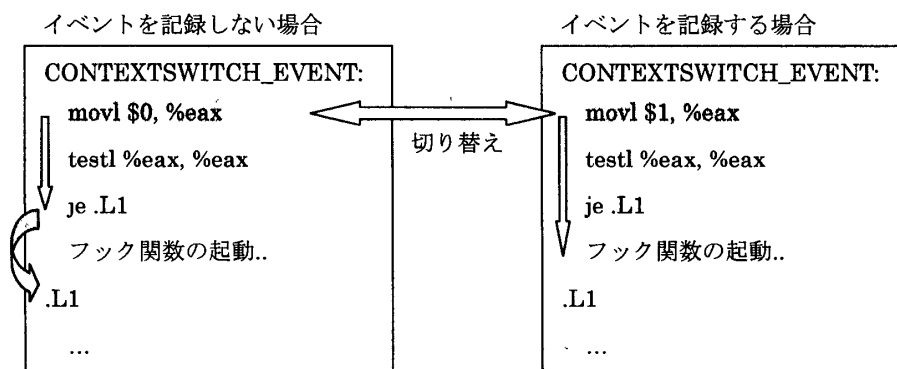


図 7. フック関数呼び出しの高速化

ように、カーネルのコードを直接書き換えることによって、フック関数の呼び出しの有無を切り替えている。直接カーネルコードを書き換えることで、フック関数の呼び出し判定の処理は不要となり、単にコードを実行するだけで良い。これにより、フック関数呼び出し判定のためのメモリ参照が不要となる。

■性能評価

上記のフックの高速化による効果を、Linux2.4.20 カーネルのコンパイルをベンチマークに評価した。その評価結果を表1に示す。
実験環境：Pentium3 800x2 Memory:512MB

OS：Linux 2.4.17

表1. トレーサの性能評価

#	カーネル	時間[s]	オーバーヘッド [%]
1	オリジナル	239.74	-
2	トレーサ性能対策無し	258.71	7.91
3	トレーサ性能対策有り	249.18	3.94

表1のとおり、ロックレスのバッファ書き込み、およびフック関数呼び出しの高速化による性能向上により、性能対策を行わない場合に比べて半分程度のオーバーヘッドへ削減できていることが分かる。しかし、本対策だけでは目標である3%以下の性能劣化を達成できていない。そこで、以下にイベントマスクテーブルの機能を利用した性能対策方法について述べる。


4. 1. 2. イベントマスクテーブルを利用した性能対策

LKSTでは、記録するイベントを選択できるイベントマスクテーブルを用意することで、デバッグ時に不要なイベントを記録せずに、性能低下を軽減する方法がある。

■イベントマスクテーブル

イベントマスクテーブルは、カーネルのイベントに対してイベント発生時に呼ばれるフック関数を動的に切り替える機能である(図8参

照)。このフック関数にNULLを指定することで、不要なイベント情報をバッファに記録しないように設定可能である。



イベント種	関数ID
Process switch	NULL
System call	1
spinlock	NULL
Interrupt	1
Network	2
printk	NULL

フック関数

図8. イベントマスクテーブル

■記録イベントの調整

記録するイベントを調整する際に問題になるのは、これから発生する障害がいかなるものかを予測できないため、どのイベントの情報を重視すべきかを事前に予測できないことである。よって、記録イベントの調整による性能向上は、信頼性維持とのトレードオフとなる。表2に Red Hat Linux の障害報告サイト Bugzilla⁷⁾にて報告されているカーネルの主要な障害に対して、デバッグに必要なイベント情報との対応を調査した結果を示す。

表2. Bugzilla のデバッグ必須情報

デバッグ必須イベント	割合
プロセス管理系	14%
メモリ管理系	29%
ハードウェア系	7%
システムコール	0%
ロック系	10%
I/O 系	40%

ここで、システムコールは0%とあるが、OSのデバッグではユーザとカーネルの界面であるシステムコール、割り込み・例外(ハードウェア系に分類)はデバッグに必須情報と判断しているため、100%必要とみなして良い。

これに対して、前述の性能評価で利用した

Linux2.4.20 カーネルのコンパイルのベンチマークにおいて、発生したイベント数の割合を表3に示す。

表3. カーネルコンパイルの統計情報

イベント分類	割合
プロセス管理系	0.44%
メモリ管理系	4.00%
ハードウェア系	9.21%
システムコール	9.40%
ロック系	76.93%
I/O系	0.03%

表3に示すとおり、発生しているイベントのうち約8割がロック系イベントであることが分かる。そこで、このロック系イベントを記録しないようにイベントマスクテーブルを設定して性能を評価した結果を表4に示す。

表4. 性能評価

#	カーネル	時間[s]	オーバーヘッド [%]
1	イベント調整無し	249.18	3.94
2	イベント調整有り	243.11	1.41

このように、記録するイベントを調整することによって、目標とする3%以下の性能劣化を達成することができた。しかし、先の表2で示したように、ロック系イベントを記録していない場合、発生するであろう障害全体の10%程度のデバッグが困難となることを考慮しておく必要がある。

4. 2. 稼動中のトレース機能の拡張

柔軟なデバッグ機能を実現するには、トレース機能が収集できる情報の種類や、イベント発生時の動作を拡張できる必要がある。しかし、機能拡張に伴ってシステムの停止が必要であれば可用性が低下する。そこで、稼動中のトレース機能の拡張を実現し、柔軟な拡張機能を実

現した。

4. 2. 1. 拡張モジュールによる機能拡張

LKSTでは、カーネル内に挿入したフックからフック関数を呼び出す。フック関数は、通常はバッファにイベントを記録する役割を持つ。このフック関数を、カーネル拡張モジュール機能を用いて他のデバッグ機能を実装したフック関数に置き換えることによって、様々なデバッグ方式を実装可能とした。図9に概要を示す。

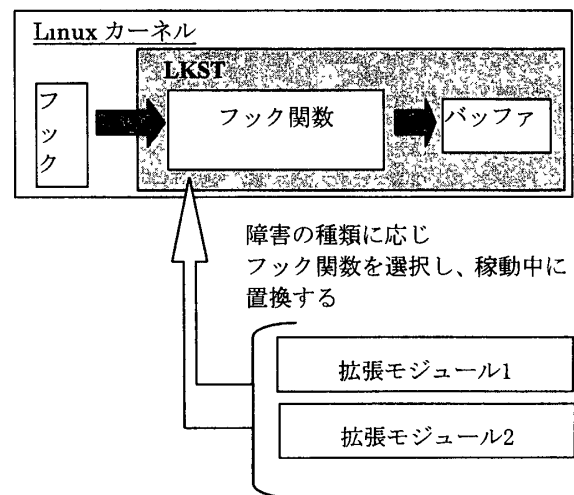


図9. 拡張モジュールを利用した機能の拡張

拡張モジュールはカーネルが動作中に追加・削除できるため、この機能によってトレースの機能を動的に追加したり、収集する情報を変更するなどの様々なデバッグが可能となる。

4. 2. 2. トレース拡張の効果

事例：不正割込みの頻発

現象：あるLinuxカーネルにおいて、/dev/zero 仮想デバイスから通常のファイルヘータ転送を行うと、転送サイズの増加とともにシステム負荷が急上昇する。ディスク入出力割込みの頻発が原因であると予測される。

解析方法：フックを通過した回数をカウントする機能を持つデバッグ用モジュールを準備し、上記現象が発生するカーネルと発生しないカ

一ネル上でそれぞれ図 10 に示すようにフック関数の入れ換えを行い、その結果を比較した。これにより、上記現象の発生するカーネルにおいては、ディスク入出力割込みが非常に多く発生しており、さらなる詳細調査の結果デバイスドライバのバグを発見した。

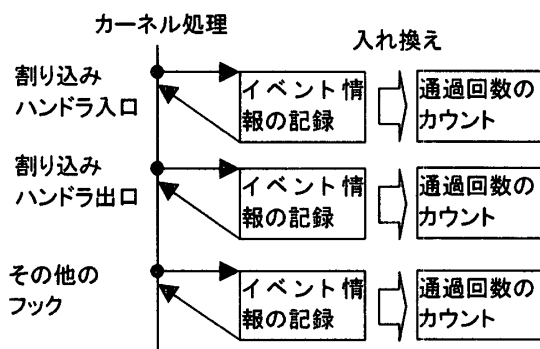


図 10. フック関数の入れ換え

ここで、デバッグプリントを利用してデバッグする方法もあるが、この場合システムリブートが必要である。拡張モジュールを利用したトレース機能の拡張では、リブートレスな機能拡張が可能となり、稼動中システムを停止することなく柔軟なシステムのデバッグが可能である。

5. まとめ

24 時間 365 日稼動を必要とするシステムにおけるデバッグ機能として、カーネルクラッシュダンプとトレーサの機能の連携によるデバッグ方法を提案した。さらに、トレース機能として実用的な性能を達成し、リブートレスな機能拡張が可能な LKST を開発することにより、稼動中システムにおけるデバッグの課題である (1) 性能を劣化させない、(2) システムの可用性を低下させない、デバッグ機能を実現した。

なお、LKST は、エンタープライズ Linux 機能強化に向けた提案活動の一環として、富士

通、日立を中心に開発を進めているデバッグツールである。

6. 今後の課題

今後の課題として、以下の 2 点が挙げられる。

- ・更なる性能-高信頼のトレードオフ解析
- ・フックの動的追加機能

参考文献

- 1) 中村哲人、畑崎恵介、芹沢一：「高信頼 OS 向けカーネル拡張モジュール機能を用いた OS デバッグの実現」SACIS2003
- 2) 杉田由美子、畑崎恵介：「Linux カーネル状態トレーサ (LKST)」Linux Kernel Conference 2002
- 3) 畑崎恵介、中村哲人、芹沢一：「システムの挙動に応じて動作の切り替えが可能なイベントトレーサ LKST の開発」FIT2002
- 4) Daniel P. Bovet, Marco Cesati「Understanding the Linux Kernel 2nd」O'REILLY
- 5) LKST :
<http://sourceforge.net/projects/lkst/>
- 6) LKCD :
<http://oss.sgi.com/projects/lkcd/>
- 7) Bugzilla
<http://bugzilla.redhat.com/bugzilla/>