C-024

組込みシステムのソフトウェア障害予兆検出に適した挙動情報収集手法の検討
A Study of Event Trace method for Embedded Software Failure Symptom

長野岳彦† 亀山達也†
Takehiko Nagano Tatsuya Kameyama

1. はじめに

組込み機器の高機能化・高性能化が進み、インターネットに接続可能なシステムが増加し、今後は、インターネットを利用したソフトウェア（以下ソフト）の構成変更が可能となる。結果、製品出荷後に導入されたソフト等の外部要因に起因した、出荷時の環境では発生しない障害の発生が懸念される。このように外部要因に起因した障害は、事前にソフトのテストで防ぐ事が困難であると考えられ、障害予防技術による対策の重要性が増してきている。本稿では、製品出荷後における障害の予防手法を提案する。また、予防を実現することに最適化した情報収集ツールを開発・評価したので、結果を報告する

2. ソフトウェア障害予兆検出方式の検討

本章では、最初にソフト障害の定義をする。次に、その障害対策方法を分類、検討し、最後にソフト障害予兆検出方式を提案する。

2.1 ソフトウェア障害の定義

組込み機器のソフト障害は、単一のソフトで発生するものから、複数のソフトの動作の組み合わせで発生するものや、外部要因によって発生するものに変化しつつある。理由は、組込み機器の高機能化、複雑化[1]と、ネットワーク接続の普及により、多くの外部システムに接続、サービスが使用可能となった点にある。結果、想定外のソフトの組み合わせで発生する障害や、完全が安全に保障されない、外部システムやデータに起因する障害が発生すると予想される。

以上より、本研究で扱うソフト障害は、複数のソフトの組み合わせで発生する障害と、ネットワークに接続されることで引き起こされるネットワーク環境依存障害の 2 点を指すものとする。

2.2 ソフトウェア障害予防

2.1 で示したソフト障害を、即時性の観点から検討し、予防方法を考案した。内容を以下に提示する。

即時性が高い障害は、障害の発生から、問題の顕在化までの時間が短く、障害の予兆を検出出来ても、対策処理が完了する保証は無い。従って、即時性の高い障害に対しては、同件障害発生を防ぐため、障害解析を実施するための情報収集と、修正が完了したソフトの再配布が必要と考える。ソフトの再配布の仕組みとしては、既に OSGi フレームワーク等があるので、障害発生時のシステム情報収集の手段が必要である。次に、即時性が低い障害は、障害の予兆が検知出来た場合、障害の内容によっては、顕在化するまでの時間情報の収集や、該当機能の縮退運転が出来る。また、障害対策が、障害の顕在化前に完了した場合は、ソフト入れ替えを実施し、運用を継続することが可能である。即ち、ソフトウェアの障害予兆検出技術の開発が求められる。

†株式会社 日立製作所 中央研究所 グリーン IT 基盤技術研究センタ
Hitachi, Ltd., Central Research Laboratory Green IT Platform Research Laboratory

以上より、本研究におけるソフトウェア障害予防とは、同一障害の再発防止、並びに障害予兆検出を用いた可用性の向上を指すものとする。

2.3 ソフトウェア障害予兆検出方式

我々は、開発時に発生するソフトウェア障害に対し、データマイニング技術を適用して問題の発生時間を検出することに成功している[2]。結果、異常の内容によっては、データマイニング技術で、顕在化する前の予兆を検出することが可能と判断し、図 1 に示す障害予兆検出方式を考案した。

基本的な考えは、大量のコンピュータリソースを必要とする学習は、ホスト側で処理し、その他情報収集と異常検出を、組込み機器で実施することである。そのため、異常を検知する識別器と、トレースモジュール及びトレースポリシー等を組込み機器上に配置した。また、トレースモジュールに関しては、障害内容によって取得情報を変更する必要が有るため、動的構成変更可能なものを採用した。

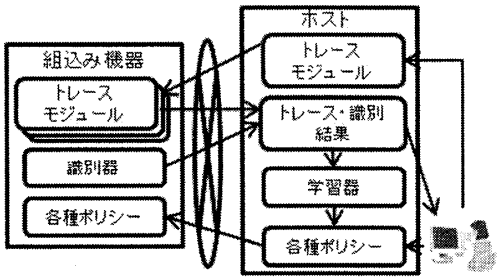

図 1.提案するソフトウェア障害予兆検出方式

3. トレース情報最適化技術の検討

本章では、2 章で提案したソフトウェア障害予兆検出システムを実現するために必要な、トレースモジュールに関して説明する。

3.1 障害予兆検出に適したトレース技術

本研究では、2.1 に述べた障害を対象としており、複合要因を切り分け可能な情報収集をする必要がある。OS はハードウェアの抽象化や、アプリの制御、割り込み処理など、システム全体の制御をしており、その挙動情報は、2.1 で示した対象の解析に活用出来る。以上の理由から、本研究では、挙動情報の収集に OS トレーサを用いる

3.2 既存トレース技術の課題

OS トレーサには、LKST[3]や LTTng[4]等がある。しかしそれらのトレーサは、メインライニングされておらず、Linux カーネルが更新される度に、トレーサの修正が必要となる。Linux カーネルの更新は、2.6.18 から 2.6.24 までのマイナーバージョンアップにおける、平均増加行数が 158,119 行と大きく、バージョンアップの度に、膨大なコ

ードの解析や、メーリングリストの追跡をしなくてはならず、OS トレーサの修正は、開発者の負担となっている。また、それらは静的にカーネルに組み込まれるので、既に組込まれているトレース内容しか収集することが出来ず、2.3 で示した、動的な取得情報の変更が出来ない。

3.3 既存技術の課題を解決する手法の提案

既存のトレーサは、3.1 で述べた課題を持つが、取得出来る情報の有用性は、既に実証されている[2]。そこで、既存のトレーサが持つトレース内容を生かしつつ、3.1 の課題を解決する手法を示す。

3.3.1 tracepointﾌﾚｰﾑﾜｰｸと Ftrace ring buffer の活用

Linux Kernel 2.6.26 より ftrace、2.6.28 より tracepoint というフレームワーク(以下 fw)が、カーネルにメインライニングされた。ftrace は、Generic ring buffer（以下 ring buffer）というデータ格納・処理系を実装している。また、tracepoint は、対象のイベントを Hook するインタフェースと、Hook に関連した処理の on/off を動的に実現する。これらの環境が揃うことで、トレーサの実装の足枷であったデータ格納処理の実装から、トレーサ開発者が解放された。結果、様々なトレーサが公開され、メインライニングされた。

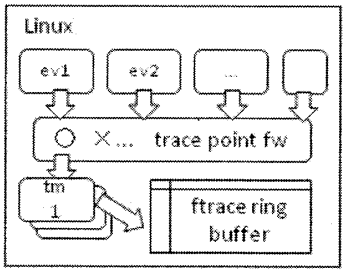

図 2.提案するトレース手法の概要

それらのトレーサは、LKST の様な、OS の挙動全般を対象としたものではなく、我々のデータマイニングを実現する上では不十分であったが、プロセス切り替え処理や、システムコールの入口等の有用な位置に Hook を登録しているため、今後我々がカーネルをメンテナンスしなくても、所定の位置から、挙動情報を収集可能となった。そこで我々は、図 2 に示すような、trace point の fw で実装された、他のイベントトレーサが埋め込んだ Hook を使い、LKST と同内容のトレースを実現し、ring buffer に記録するトレースモジュール(tm)を開発し、2.3 に示した予兆検出方式を実現することにした。

4. 開発したトレーサの開発と評価

我々は提案手法によって実装したトレースモジュールを Linux kernel 2.6.23 に実装し、関連して tracepoint fw、ring buffer を Linux Kernel 2.6.23 向けにバックポートした。

実装したトレーサの与える負荷の影響を調べるため、CPU 使用率の測定と、ベンチマークツールを用いて OS の処理に与える影響を測定した。CPU 使用率は、Snap Gear 社の Greg Ungerer 氏が作成した OSS ツール cpu.c を使い測定した。ベンチマークツールは、Unix 系 OS の性能を多角的に検証可能な Unixbench[5]を使用した。

比較対象は、トレーサを組込んでいない linux-2.6.23(vanilla)と、既存トレース技術を代表して LKST との比較を実施した。測定したハードウェアの環境は、表 1 の通りである。情報系組込み機器を想定し、比較的小型な機器を想定した 1 と、大型なものを想定した 2 を用意した。また、上記環境上で、実システム運用を想定し、http サーバ(thttp)を動作させ、ネットワーク外部から、Apache Bench[6] を使い、対象機器に対し、CPU 負荷を 0%,30%,50%となる様調整をしながら、測定を実施した。

表 1. ベンチマーク測定環境（ハードウェア）

1	2
Open BlockS 266 CPU:PowerPC405GPr 266MHz Memory: 128MB	PC CPU:Intel Celeron T3000 (1 core)1.8GHz Memory:2GB

まず、CPU 負荷の測定結果を表 2 に示す。トレーサの負荷は、OS の処理として扱われるので、system の値が、各負荷でどの程度増加したかを測定した。以下に見るとおり、PC では 1%以下、OpenBlockS では 5%程度と、実用上問題無いと考えられる。しかし、既存手法（LKST）よりは、負荷が高いという結果を得た。

表 2 CPU 使用率(system)増加分(対 vanilla 単位は%)

1.PC				2.Open BlockS			
負荷 30% 時		負荷 50%時		負荷 30%時		負荷 50%時	
LKST	提案	LKST	提案	LKST	提案	LKST	提案
+0.56	+0.23	+0.70	+0.06	+0.87	+5.07	+0.40	+5.18

次に、OS 自体の負荷が高い状態における影響を、Unix Bench の Final Score(複数指標を統合した結果)で評価した。結果を表 3 に示す。この表は、CPU 負荷 0%の場合に比べ、スコアが劣化した比率を示している。各 CPU 負荷において、提案手法は LKST と比べ,OBS では平均 10.63%,PC では 13.13%劣化しており、OS の負荷が高騰した場合は、LKST より副作用が大きいという結果を得た。

表 3 Unixbench(finalscore)の劣化率(単位は%)

	1.Open BlockS			2.PC		
	0%	30%	50%	0%	30%	50%
lkst	21.86	32.04	56.74	20.20	41.64	42.43
提案	35.33	43.56	63.62	37.11	51.32	55.23

5. まとめと課題

本稿では、ソフトウェアの障害予兆検出を実現する一方式を示した。また、障害予兆の診断に適し、挙動情報収集手法の一例を開発し、評価し、実用上問題無い CPU 負荷で動作することを示した。

今後の課題は、以下 2 点である。

1. 提案手法の高負荷要因の明確化と、対策案の提示
2. 提案手法の予兆検出に関する評価手法の定式化

参考文献

[1]高田広章“組込みシステム開発技術の現状と展望”,情報処理学会論文誌,Vol.42,No.4(2001)
[2]長野岳彦ら,“長時間トレース技術を用いた組込みソフトウェアの開発効率向上に関する検討”,FIT2009
[3]畑崎恵介,中村哲人,芹沢一,“稼動中システムのデバッグを考慮した OS デバッグ機能”情報処理学会研究報告.[システムソフトウェアとオペレーティングシステム]pp.33-39.2003
[4]Mathieu Desnoyers et al , “The LTTng tracer; A low impact performance and behavior monitor for GNU/Linux”,OLS2006 Proceedings (2006)
[5]http://www.tux.org/pub/tux/niemi/unixbench/
[6]http://httpd.apache.org/docs/2.0/programs/ab.html