Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

組込みシステム向け障害解析環境の効率改善

長野岳彦[†1][†3]　小口琢夫[†1]　吉岡信和[†2]　田原康之[†3]　大須賀昭彦[†3]

概要：デジタルテレビや携帯電話，カーナビゲーションシステムに代表される組込みシステムの高機能化・複雑化が進み，開発工数が増加している．一方でメーカ間のシェア競争は激化し，開発コストの削減が望まれている．組込みシステムの障害は，ハードウェアとソフトウェアが複数組み合わさって動作するため，タイミングに依存し，再現性が低く障害解析に必要な情報を取るために時間がかかるといった問題や，システム全体を解析対象とするため，トレース結果の解析に時間がかかるといった問題を抱えている．そこで我々は，上記問題を解決するための障害解析環境を提案する．提案環境は再現性の低い障害の解析情報を確実に取得するための長時間トレース機能と，トレース結果の解析効率化を狙うボトルネック解析機能，解析情報理解容易化機能の 3 機能からなる．この障害解析環境はデジタルテレビ HR-01 シリーズの開発から順次 6 製品に適用され，現在も一部製品開発に適用されている．また，一例としてトレース容量を 10MB から 190GB へ増加し，障害解析時間を 94.8%削減する結果を得た．これらの結果について報告する．

キーワード：障害解析，デバッグ，組み込みシステム，トレース

Improving the efficiency of failure analysis environment for embedded systems

TAKEHIKO NAGANO[†1][†3]　TAKUO KOGUCHI[†1]　NOBUKAZU　YOSHIOKA[†2]
YASUYUKI　TAHARA[†3]　AKIHIKO　OHSUGA[†3]

Abstract: Embedded systems work with a combination of hardware and software. Therefore, the occurrence of a failure depends on the timing, and the reproducibility is low. As a result, there is a problem that it takes time to obtain the information necessary for failure analysis. In addition, the entire system will be analyzed. Therefore, there is a problem that it takes time to analyze the trace result. Therefore, we propose a failure analysis environment consisting of three functions to solve the above problem. They have the following three functions.1) The proposed environment has a long-time trace function to reliably acquire analysis information of failures with low reproducibility.2) Bottleneck analysis function that aims to improve the analysis efficiency of trace results, 3) Function that facilitates understanding of analysis information

This failure analysis environment has been applied to six products in sequence from the development of the digital television HR-01 series. And above environment is still being applied to some product development. Also, as an example, the trace capacity was increased from 10MB to 190GB, and the failure analysis time was reduced by 94.8%. We report these results.

Keywords: Failure analysis, Debug, Embedded system, Trace

1. はじめに

組込みシステムの開発にはさまざまなハードウェア，ソフトウェアの組み合わせが用いられる．製品の使用目的，要求される性能，かけられるコストなど要求がさまざまなためである．また近年のハードウェア高性能化に伴い，ソフトウェアによる機能実装が増加している．結果，ソフトウェアの開発規模は増加の一途をたどっている．

このようなハードウェアの高性能化及びソフトウェアによる機能実装の増加により，組込みシステムでは，これまでのような特別なハードウェアを前提としたソフトウェアの実行だけでなく，OS，ミドルウェアの実行や，機能を実現するアプリケーション（以降アプリ）が複数まとめて動作する機会が増えている．結果これまで発生しなかったような，アプリ間のリソース競合による障害などが発生するようになり，その影響評価や実行順序の妥当性検証に工数を要することが増えてきた[1].

このようにソフトウェア開発工数は増加しているものの，製品コスト削減の要求から，開発期間の削減が求められている[2]．開発期間の削減には，多数の開発チームによる並列開発や，テスト工数の削減，デバッグの効率化が必要である．

我々は，この中でデバッグの効率化に着目し，デバッグツールを見直すことにした．デバッグツールには，動作中のソフトウェアを制御しながら解析をする対話型デバッグツールや，プログラムの動作情報を記録し，動作終了後に挙動を解析するトレースツールや，可視化ツールなどがある[3]．ここで，これらのツールを組み合わせ，障害解析全

†1 (株)日立製作所 研究開発グループ システムイノベーションセンタ
Hitachi Ltd., Research & Development Group, Center for Technology Innovation
†2 早稲田大学 理工学術院総合研究所
Research Institute for Science and Engineering, Waseda University
†3 国立大学法人 電気通信大学 大学院情報理工学研究科
Graduate School of Informatics and Engineering, The University of Electro-Communications.

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

体を効率良く進める方法を検討し，評価することにした．

本論文では，デジタルテレビ（以降 DTV）のアプリ開発を対象に, ソフトウェアの障害解析における課題を抽出し，それに対する改善手法を提案，提案した内容を実装し，DTV を中心とする組込みシステム 6 製品に適用した結果について報告する．

以降，第 2 章では組込みシステム向け障害解析環境の課題について述べる．第 3 章では 2 章で述べた課題を解決するための，障害解析環境の効率改善に必要な機能の要件について述べる．第 4 章では，3 章で述べた要件を満足する各機能の実装について述べる．第 5 章では，第 4 章に記載した内容の評価と，複数の組込み製品に対して適用した実績ついて述べる．第 6 章では，結論と今後の課題について述べる

2. 組込みシステム向け障害解析環境の課題

これまで日立製作所では, DTV やビデオカメラなどを中心とした情報系組込みシステムの開発を多数行ってきた．その中でも特に，組込みシステムの Linux*化，ネットワーク対応を契機に，ソフトウェアの複雑さは増し，障害解析に必要な時間は増加している．

そこで，我々はこれまで開発してきた DTV とビデオカメラの開発実績をもとに，障害解析におけるどのような作業が開発工数に影響を与えているかを調査した．その結果は，以下の通りであった．

1)障害の再現性が低く，再現をさせるまでに時間がかかる
2)システム全体の膨大なトレース結果を解析するのに時間がかかる．
　2-1)試行錯誤しながらログを取る
　2-2)ログをとれる時間が多いが，情報量が膨大であり，解析に時間がかかる

それぞれの具体的な根拠について説明する．

2.1 再現性の低い障害

組込みシステムの開発では，障害の再現性が低く，障害解析が進まないケースが散見される．例えば，我々が 2007 年に製品開発をしたデジタルビデオカメラ（以降 BD-CAM）の障害 2600 件のうち，19 件（0.7%）の障害が 1 週間（5 営業日）以上の再現がしなかった「再現性の低い障害」に該当する障害であった．

この 19 件は再現性のある障害に対し，平均対策期間でも長い時間を必要とした．再現性のある障害の対策期間が平均 10.7 日であったのに対し，再現性の低い障害の対策期間は平均 19.8 日と，1.85 倍の対策期間を要した．

我々が再現性の低い障害の内容を解析した結果，それらはアプリレイヤ以外の OS，デバイスドライバ以下のレイヤに原因があった．そのため，障害を発生させるためには担当者の開発したアプリ以外のシステムの条件を揃える必要があり，再現が困難であった．このため，解析に必要な情報の取得に時間がかかり，解析のボトルネックとなっていた．以上より，以下の課題を得た．

課題 1:再現性の低い障害の情報を確実に記録できるようにする必要がある．

2.2 トレース結果の解析効率化

組込みシステムの障害解析においては，我々は 2006 年の DTV 開発のタイミングで，OS トレーサである LKST(Linux Kernel State Tracer)[4]を導入することで，2.1 に示したアプリレイヤ以外で発生する障害などに対応可能にした．一方で, OS トレーサはアプリ開発者への十分な普及はしなかった．

その原因を調査するため, DTV 開発チーム及び BD-CAM 開発チームに対し, 使用しない理由をヒアリングした結果, 以下の様な回答を得た．

a)LKST のトレース結果が，膨大でありどこから見てよいかわからない
b)LKST のトレース結果は膨大である一方，長時間のトレースが出来ないため, 2.1 に述べたような再現性の低い障害の解析情報収集に時間がかかる
c)障害情報収集後に，アプリ名称, 割り込み名称，システムコール名称といった解析に必要な情報を引き当てる必要があるが，その作業が難しいうえ，膨大

順にヒアリング結果について説明する．a)については，LKST はカーネルのメモリ領域にトレース結果を保存し，それをテスト完了後にコマンドを使い抜き出す．ここで組込みシステムのメモリ領域のうち，LKST のトレース結果保存に使える領域は多くて 10MB 程度であった．これらのメモリを使い, OS のトレースを取ると，システムの動作状況に依存するが，10 秒弱，5 万行～6 万行程度のログが出力される．この量は，解析担当者にとっては非常に膨大である．以上より，以下の課題を得た．

課題 2:アプリ開発者からすると，LKST のトレース結果が膨大．そのため，解析対象を絞り込む必要がある．

次に b)については，解析するデータ量は膨大である一方で, 10 秒という時間の中で障害を再現させることは難しく, 課題 1 同様，再現性の低い障害の情報でも確実に記録できるようにする必要がある. 最後に c)については, DTV は 194 タスク，BD-CAM は 78 タスクが同時に実行され，id で管理されている．しかし，大半のタスクは分析には不要なものであり，解析に必要なトレース結果の名前解決をして，人手で抽出する必要があった．同様に，アプリの動作のトリガとなるハードウェアの情報や，OS とのやり取りをおこなうシステムコールの情報も，システム上は名前ではなく，システムの動作時に付与される id で管理されるため,

* Linux®は Linus Torvalds の日本及びその他の国における登録商法または商標です.

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

人間が解析するには，名前解決をしないと理解が難しい．更には，文字の情報としてトレースを直接解析することは困難で，可視化など理解容易化が必要である．以上より，以下課題を得た．

課題 3:開発者が理解しやすい形で解析情報を提供する必要がある．

2.3 関連研究

組込みシステムの障害解析においては，製品が市場投入されるまでの時間が問題となっており，デバッグフェイズの最適化が必要である．そこで実行トレースを用いた分析が強力であることは既にわかっており[5]，多くの研究がされている．

例えば，トレース分析結果を可視化する研究は様々あり，Java で実装されたプログラムの実行履歴を可視化する JIVE をはじめ[6], マルチプロセッサ環境で動作するソフトウェアデバッグ用のトレース可視化ツール TLV[7]，リアルタイム性を持つ組込み機器向けに開発されたオープンソースの可視化ツールである Timedoctor[8]など，多くの開発と報告がされている．また，組込みシステム向けを含め多くのトレースツールや[9][10][11][12]，プロファイルツールなど解析用の報告もされている[13][14]．また，大量に取得したデータの効率化については，パターン認識を用いて周期性のある振る舞いの情報を抽出する研究がおこなわれている[5].

一方で，これらの報告は，トレース自体，可視化自体を対象に進めているものが多いが，我々が課題に挙げている再現性の低い障害の解決には，十分に考慮されていない．例えば，トレースを取得する点においては，再現性の低い障害が発生した際のトレース情報を，確実にとらえなければならない．またその際に取得した膨大な情報を効率良く解決しなくてはならないが, その情報は周期性が無いため，検出するには別のアプローチの検討が必要と考えらえる．

そこで本研究では，これらの先行研究を踏まえ，効率良く再現性の低い障害を解決する機能について検討・評価する

3. 障害解析環境の効率改善機能の要件

2 章で示した課題 1〜3 を解決する，以下に示す要件 R1〜R3 を満足するソフトウェアを開発することを目標とする

R1:長時間トレース機能の実現

R2:膨大なトレース結果から解析対象を絞り込む機能の実現

R3：システム全体のトレース結果を取得し，アプリ開発者でもわかる形式に変換する機能の実現

R3-1：時系列上にプロセス毎にトレース結果を表示できる機能

R3-2:時系列上にハードウェアからの割り込み処理の開始契機を割り込み信号ごとに表示できる機能

R3-3:システムコールの発行タイミングをシステムコールごとに記録できる機能

以降，本章ではそれぞれの詳細について説明する．

3.1 長時間トレース機能の実現

2.1 で示した課題 1 を解決するための機能である．再現性の低い障害の解析を効率良く解析するためには，偶然障害が再現した際に確実に情報を記録していることが必要である．また，2.2 の 2)に示した通り，LKST は，カーネルが確保したメモリ領域の一部にトレースを記録する．そのためハードウェアリソースに制限のある組込みシステムでは，記録時間が少なくなり再現性の低いバグの記録や，長時間の耐久テストなどでは仕様できない．そこでトレース結果をメモリ領域に記録せず，代替装置に記録できる機能の実現を目標とする．

3.2 膨大なトレース結果から解析対象を絞り込む機能の実現

2.2 の 1)に示した通り， OS トレースは，OS を介して処理される全てのプロセスやハードウェアとのやり取りの情報に関係するイベント情報を取得する．そのため，取得できる情報の量は膨大であり，問題点を絞り込まないと解析の効率が下がる．そのため，実行時に CPU を多く利用しているものといった観点から解析対象を絞り込める機能の実現を目標とする．

3.3 アプリ開発者が理解しやすい形式への変換

2.2 の 3)に示した通り，LKST の結果は，OS で実行されるイベントについてトレースを取る．その解析の簡略化のため，以下に述べる 3 つの機能の実現を目標とする．

3.3.1 OS トレースの結果をアプリごとに分類できること

OS 上でアプリは，プロセスの単位で実行される．ここで OS トレースの結果は，プロセスの切り替えや，システムコールの発行，セマフォの確保といったアプリの動作に関係する様々な情報を取得している．それらの結果は，pid などプロセスを識別する単位で記録されている．一方，プロセスは OS がプロセスを起動する際に OS によって pid が振られるため，システムテストなどトレースを取得するたびに pid の番号が変わってしまう．そこでテスト実行時における pid の情報をテスト実行時に取得し，その結果を解析時に突合して解析を可能にすることを目標とする．また，pid の情報とプロセス名称の情報を突合出来るように，/proc/[pid]/以下の情報を一括取得し，pid 番号とプロセス名を突合して解析を可能とすることを目標とする．最後に，取得した各プロセスの情報から，CPU を占有している時間を把握可能にすることで，プロセス間のやりとりなどが視覚的に把握可能となるため，その実現を目標とする．

3.3.2 ハードウェアの制御とアプリの動作の関係を対応づけられること

組込みシステムは，ハードウェアを用いた制御をおこな

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

うことが多く，ハードウェアからの動作指示を契機にアプリが起動することがよくある．そのため，割り込みの情報とアプリの動作の因果関係を把握出来る必要がある．そのため，割り込み情報をプロセスの情報と同じ時間軸上で解析（一緒に解析）できる機能の実現を目標とする．

3.3.3 システムコールの発行とアプリの動作の関係を対応づけられること

組込みシステムは OS を介してハードウェアとやり取りをすることが多い．そのため，アプリからシステムコールを発行してハードウェアを制御することや，やり取りをすることがよくある．そのため，システムコールの発行とアプリの動作の因果関係を把握する必要がある.そのため，システムコール発行情報をプロセスの情報と同じ時間軸上で解析（一緒に解析）できる機能の実現を目標とする．

4. 障害解析環境の効率改善機能の設計

4.1 障害解析環境の効率改善機能の全体像

今回開発する機能群の全体像を，以下図 1 に示す．

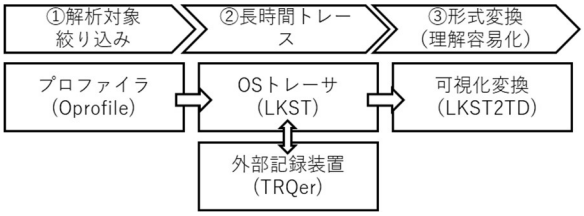


図 1 今回開発する機能群の全体像

今回開発する内容は，要件 R1 を満足する②長時間トレース機能，要件 R2 を満足する①解析対象絞り込み機能，要件 R3 を満足する③形式変換機能の 3 つである．これらの機能の設計について，以降説明する．

4.2 長時間トレース機能の実現

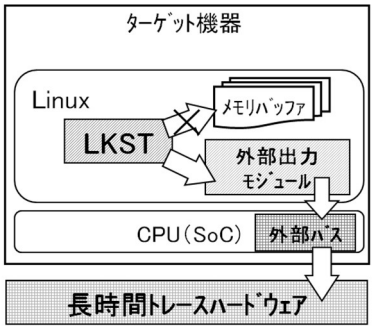
今回の長時間トレース機能は，基本 OS トレーサのイベントを，可能な限り長時間トレース出来る機能を再利用する[15]．解析に必要な情報は LKST で取得するものとし，LKST の記録を外部に保存する仕掛けを考える．

従来の LKST において長時間のトレースをするには，メモリ上に複数面のバッファを確保し，そのうちの 1 面のバッファ領域に対しイベントが発生する毎に内容を記録し，そのバッファを使い切った後バッファを切り替え元のバッファの記録結果を外部媒体にファイルとして出力し，長時間トレースを実現している．しかし，組込みシステムに搭載されるメモリは容量に制限がある．また，出力先のストレージデバイスも無い場合が多い．更には，ファイルへの出力は，ファイル格納を格納するためのストレージデバイスへの書き込み負荷が，テスト対象のシステムの動作を変更してしまう可能性がある．

そこで，提案手法では，システムが外部にデータを出力する外部出力バスを持っていることを前提とし，外部バス経由で記憶容量が大きい外部記録装置に低負荷で OS の挙動情報を記録する方式を提案する．

LKST のトレース結果を送出するための外部出力モジュールをデバイスドライバとして実装し，挙動情報を LKST のバッファ領域外に出力可能にした．この出力モジュールには，LKST のイベントハンドラを用い，トレース内容をカーネルに記録する処理を Hook して，本来メモリに記録する 32bit×4 のデータ列を 16bit×8 のデータ列に分解し，16bit ずつ外部バスに出力する．外部バスには，HDD を保有する長時間トレースハードウェア（今回は横河デジタルコンピュータの TRQer） を接続し，バスにイベントの出力が流れる度，長時間トレースハードウェアに記録する方式とした．

図 2 長時間トレース機能の構成



4.3 解析対象絞り込み機能

2.2 に示した通り，本研究が対象とする組込みシステムのタスク数は多いため，トレース結果を解析する際に全てを解析することは困難である．そこで解析を絞り込む方法を検討した．1)対象とするタスク数を制限する方式，2)対象とする時間を制限する方式の 2 点である．

ここで，これまでの開発において LKST の仕組み上，カーネル内部で確保できるメモリの制約による時間的な記録制限を受けることが多かった．しかし，以下図 3 に示すようなタイマ未クリアにおけるシステムリセットが発生する障害の場合，タスク 1 が割り込み処理を停止してから，リセット用タイマをクリアするまでの時間が長い場合，時間で解析制限がかかると，原因箇所のトレース結果を解析が出来ない可能性が有ることがわかっていた．

同様に障害の原因発生時刻と顕在化時刻の間隔が長い事例には，メモリーリークや排他処理，優先度変換処理など多数考えられる．そこで今回は，対象とするタスク数を制限する方式を検討することにした．そこで我々は，CPU の使用率に着目し，動作していないタスクや，動作頻度が低いタスクなどを取り除くためにプロファイラを用いることにした．

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

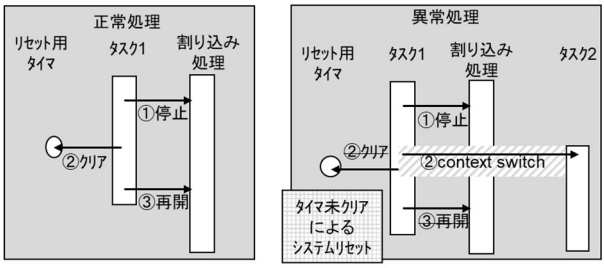


図 3　時間による制限をかけにくい障害の例

本研究の取り組み時点では，OProfile[14]が Linux では標準的に使われていたため，本報告では OProfile を使った絞りこみを行う．

OProfile は Linux 向けプロファイラの一種であり，1 つの OS からなるシステム上で動作する，全てのプログラム（OS，アプリ）の，プロファイル情報を取得するプロファイラである．OProfile はハードウェアタイマやソフトウェアタイマを使用し周期的に割り込みを発生させ，その割り込みによって，エラープログラムカウンタ（以下 EPC）レジスタに退避されたプログラムカウンタ（以下 PC）の値を取得する．更に，取得した PC の値を，アプリ，関数などの単位で統計的に解析し，ユーザに提示する．ユーザは OProfile の解析結果を基に，実行される頻度の高いアプリや関数を知ることができる．また，分析データをファイルに保存可能であるため，長時間のデータ採集に適応可能なこと，ソースコードに改変を加えることなくプロファイリングが可能という特徴を持つ．

OProfile のデータ収集機能は，図 4 に示すデータをサンプリングする割り込みハンドラ部分からなる OProfile モジュールと，ユーザインターフェースであるコマンドと，デーモンプログラムとして常駐し，カーネルからプロファイル結果を取得し，プロファイル結果がどのプログラムのものかを引き当ててデータとして保存する OProfile デーモンから成る．

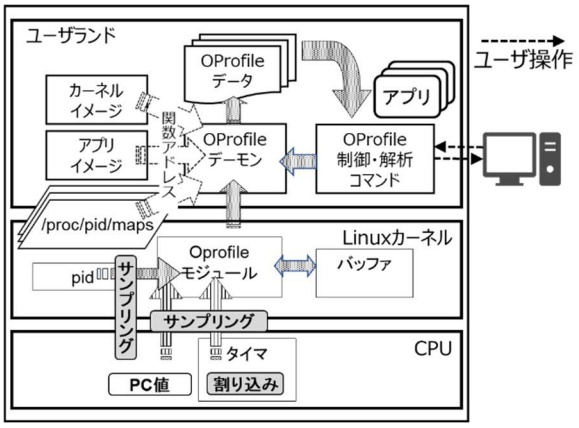


図 4　OProfile 概要

OProfile は DTV 開発時には MIPS アーキテクチャ向け MontaVista Linux3.1(Kernel 2.4)に対応していなかったため，必要な機能を移植することにした[16].

当時既に Kernel2.6 の MIPS 向けには OProfile が開発されていた．一方で，i386 及び ia64 アーキテクチャ向けには，2.4x,2.6x ともに OProfile が提供されていたため，我々は i386/ia64 向けの 2.4 向け OProfile を 2.6 向けの OProfile と比較し，カーネルバージョンの違いでどのようなプログラムの差分があるかを調査した．その結果，図 4 に示した OProfile モジュール内において，3 つの実装が異なることがわかった．1 つ目は，タイマユニット関連で CPU のキャッシュヒットミスの検出方法の実装，2 つ目は割り込み間隔の設定に伴うハードウェアタイマの設定の実装，3 つ目はアプリがライブラリをダイナミックリンクする際に発生するシステムコールの情報を収集するために，システムコールをフックして情報を収集している箇所の実装であった．我々は，上記 3 か所を 2.6 の MIPS 向け OProfile から移植した[16]．また，2.4 系のカーネルと，2.6 系のカーネルでは，デバイスドライバの I/F が変更されていることから，そこの部分の対応も併せて行った．

4.4 形式変換機能

形式変換では，TRQer に保存された LKST の記録情報から，3.3 に示した要件で出力する．今回は，プロセスや割り込み，システムコールごとに名前解決をしつつ，OSS の可視化ツール TimeDoctor で出力できる変換を行う．

LKST で取得出来るイベントは以下表 1 に示す 118 のイベントからなる．

表 1　LKST で取得できるイベント

#	Category	Event 数	備考
1	Process management	13	PROCESS_CONTEXTSWITCH, WAKEUP,SIGSEND など
2	Interrupts	10	INT_HARDWARE_ENTRY,TASKLETHI_ENTRY など
3	Exceptions	6	EXCEPTION_ENTRY,EXIT など
4	System calls	2	SYSCALL_ENTRY,EXIT など
5	Memory Management	15	MEM_SWAPOUT,SWAPIN,MALLOC など
6	Networking	5	NET_PKTSEND,PKTRECV など
7	SysV IPC	11	SYSV_IPC_SEMOP など
8	Locks	8	LK_SPINLOCK,WRLLOCK など
9	Timer	5	TIMER_RUN,ADD など
10	Oops	1	OOPS_PGFAULT
12	Others	4	O_PORTIN,PORTOUT など
13	Page	18	PAGE_ALLOC_ENTER など
14	IPv4	5	NET_V4RTIN_ENTER など
15	LKST	15	LKST_INIT,BUFF_SHIFT など
合計		118	

ここで，今回#1 に示したプロセス管理のイベント，#2 に示した割り込み関係のイベント，#4 に示したシステムコールイベント，その他のイベントに分け，それぞれプロセス名，割り込み名を引き当て，可視化する．それを実現するため，トレース結果を取得する際に，図 5 に示す結果取得処理の順でトレース結果及び関連情報を取得する．

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

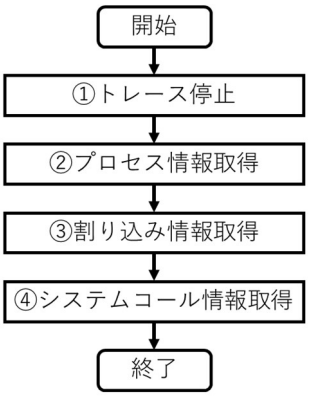


図 5　結果取得処理

まず①トレース停止処理で LKST を停止する．次に②プロセス情報取得処理で/proc/[pid]/以下の情報（例えば cmdline 情報）などを取得し，pid とプロセス名を対応付けたファイルを出力する．次に③割り込み情報取得処理では /proc/interrupts 情報を取得し, 割り込み番号と割り込み名称を対応付けたファイルを出力する．最後に④システムコール情報取得処理では，/asm/unistd.h 情報を取得し，システムコール番号とシステムコール名称を対応付けたファイルを出力し，終了する．以上の処理により，解析に必要な情報を取得する．

上記した結果取得処理の後，TRQer から取得した LKST トレース結果と合わせ図 6 に示した形式変換処理を行う．

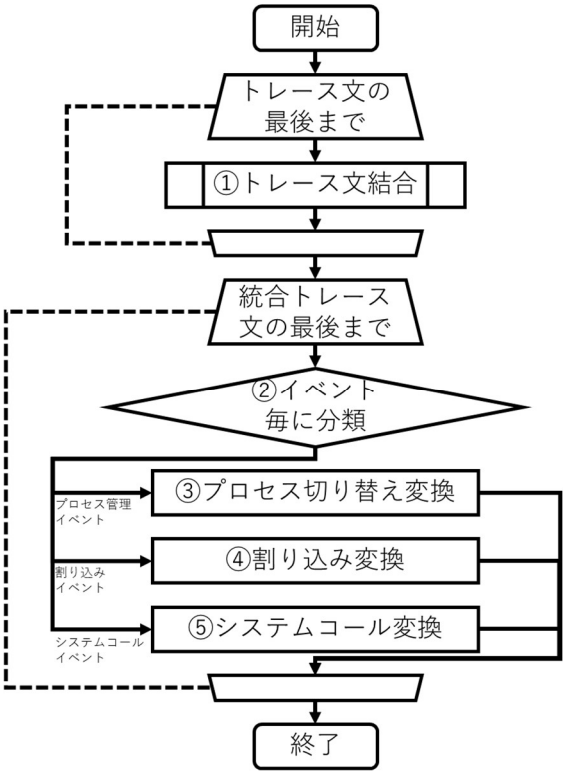


図 6　形式変換処理

まず，①のトレース文結合処理で，TRQer の仕様に合わせて保存されている LKST 情報を，元の LKST 形式の情報に変換する．具体的には, LKST1 イベントの情報が表 2 に示す 10 個の TRQer メッセージで出力される．10 個を一つのトレース文に統合する処理を行う．

表 2 TRQer に保存したトレース結果

#	TRQer ID	内容
1	A	LKST event ID
2	1	第 1 パラメータ上位 16bit
3	2	第 1 パラメータ下位 16bit
4	3	第 2 パラメータ上位 16bit
5	4	第 2 パラメータ下位 16bit
6	5	第 3 パラメータ上位 16bit
7	6	第 3 パラメータ下位 16bit
8	7	第 4 パラメータ上位 16bit
9	8	第 4 パラメータ下位 16bit
10	9	イベント発生時刻

次に，統合された各トレース文を 1 行ずつ取り込み，②イベントのマッチングをとり，各イベントに合わせた処理を行う．プロセス管理イベントの場合，③プロセス切り替え処理に遷移し，結果取得処理で取得したプロセス情報を用いてプロセス ID 毎にテーブルを作り，プロセスが実行状態・停止状態に遷移したかを時間情報と合わせて保存する．同様に割り込みイベントの場合，④割り込み変換処理に遷移し，結果取得処理で取得した割り込み情報を用いて割り込み番号毎にテーブルを作り，割り込み発生時刻と終了時刻, 割り込まれたタスク名を合わせて保存する. また，システムコールイベントの場合，⑤システムコール変換処理に遷移し, 結果取得処理で取得したシステムコール ID 毎にテーブルを作り，システムコールの発行時刻とシステムコール名称を記録する．これらの処理を，全てのトレース文が終了するまで行い，可視化ツールのフォーマットにあわせて出力する．

5. 実装と評価

5.1　長時間トレース機能

4.2 で提案した長時間トレースシステムの実用性を評価するため，Apache Bench を用いて，外部から評価対象に対し負荷をかけ，性能と記録時間を測定した．評価項目は CPU の使用率と Apache Bench によるテスト実行時間を用いた．

5.1.1 評価環境

組込みシステム（Atmark Techo 社 Armadillo†-300）と負荷を外部から与える PC（OS:Linux）をローカルネットワークに接続し，組込みシステム上で，web サーバ（thttp）を動作させ，PC から Apache‡ Bench を使用し，組込みシステム上の web サーバに大量のアクセスを発行，負荷を与える．使用したコマンドラインは“ab –n 30000 –c5 アクセス先

† Armadillo は株式会社アットマークテクノの登録商標です

‡ Apache は Apache Software Foundation の登録商標または商標です

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

URL”である. CPU の使用率は, Snap Gear 社の Greg Ungerer 氏が OSS で公開している cpu.c を用いた．cpu.c は/proc/stat 以下にある CPU の動作状況から，システム時間，ユーザ時間等を算出するツールである．

5.1.2 評価結果

まず，CPU 使用率による負荷の評価結果を，以下表 3 に示す．

表 3 CPU 負荷計測結果

	CPU 使用率	内 System
LKST 無	92.00%	36.80%
LKST ﾒﾓﾘ記録	92.45%	42.32%
LKST ﾌｧｲﾙ記録	95.02%	34.47%
提案環境	92.23%	56.94%

Apache Bench による負荷が高いため全体的に CPU 使用率は高いが，従来手法である LKST のトレース結果をファイルに記録する手法の CPU 負荷が最も高く，3.02%上昇している．それに対し，長時間トレースを用いる場合 0.23%の上昇と，それ程 CPU 負荷に影響が出てない．しかし，CPU 使用率のうち，OS が CPU を使用した結果であるシステムの項目を見ると，長時間トレースが，LKST 無しの場合に比べ 20.14%上昇している. 理由は外部バスにトレース結果を出力するカーネルモジュールが頻繁に動作しているためである．従来手法のシステムの項目が低い理由は，メモリからファイルに結果を出力する際，ユーザランドのデーモンプログラムが頻繁に動作しているためである．

次にトレース時間の評価結果について述べる．上記環境でヒートランをした結果，トレースデータを 190GB，実行時間で約 105 時間程度の記録を確認出来た．また，実際の障害としては，1000 秒程度のトレース結果を用いてメモリーリークの解析を行ったのが最長の結果となった．

5.2 解析対象絞り込み機能

4.3 で提案した解析対象絞り込み機能をテストするため，DTVHR-01 にて発生した，図 3 に示すタイマ制御処理障害の解析を通して評価した．その結果を以下表 4 に示す．

ここで示す通り，実際に動作していたタスク数は 194 あったが，OProfile にて障害発生時刻周辺で CPU を占有していたタスクを上位 10 まで絞り込み，解析を行った．これにより，ソースコードのチェックする規模も 248 万ステップから 12 万 8 千ステップ程度まで削減することが出来，解析範囲を約 1/20 まで減らすことが出来た．

表 4 絞り込みの結果

	調査タスク数	調査ステップ数
従来	194	2,486,836
成果	10	128,180

5.3 形式変換機能

4.4 で提案した形式変換機能を，DTVHR-01 と，トレースログ可視化ツールである TimeDoctor[8]を使い評価した，その結果例を以下図 7 を用いて示す．

形式変換機能では図 6 に示した通り，③プロセス切り替え変換，④割り込み変換，⑤システムコール変換を行っている．それぞれの変換結果は図 7 の①～④に出力される．図 7 の①は，カレントプロセスの切り替えを TASK としてタイムライン上にプロセス名と対応付けて表示している．②は割り込みの開始・終了を ISR としてタイムライン上に割り込み名称と対応付けて表示している，③は LKST イベントのマーキングであり，例えば④で表示するシステムコールイベントの発生タイミングがわかる．④はシステムコールの開始と終了を AGENT にシステムコールとしてタイムライン上に表示している．システムコールを発行したプロセス名と一緒に開始から終了までの時間を把握することができる．このように，トレース結果を形式変換してアプリ開発者が理解しやすい形式へ変換することができた．

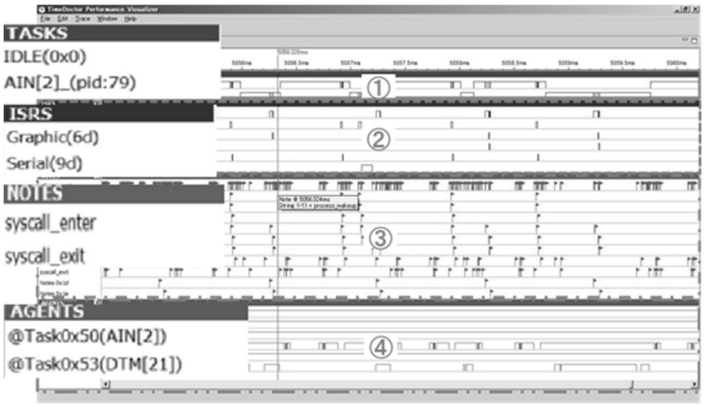


図 7 形式変換結果例

5.4 提案した障害解析環境の製品開発適用結果と考察

提案した環境は，当時の DTV 開発（日本・欧州・北米）時の解析困難な障害（5 営業日以内に担当者が解決出来なかった障害）60 件に適用することが出来た．

次に，そのうちタイマ処理制御障害を例に提案手法においてどのような工数短縮結果があったかを表 5 示す．実際の対策工数は，6.09 日短縮することが出来た．短縮出来た箇所はトレース取得及び解析の日数で, 7.5 営業日を絞り込み及びトレース取得・解析の 1.41 日へ短縮することが出来た．一方で絞り込みのための 0.41 日分は工数が増加した．

表 5 工数短縮結果

	所要時間	内訳
従来	8 日	ﾄﾚｰｽ取得・解析 7.5 日，開発・テスト 0.5 日
成果	1.91 日	絞り込み 0.41 日，ﾄﾚｰｽ取得・解析 1 日，開発・ﾃｽﾄ 0.5 日

Vol.2021-MBL-100 No.5
Vol.2021-UBI-71 No.5
Vol.2021-CDS-32 No.5
Vol.2021-ASD-21 No.5
2021/9/2

次にこれら環境を情報系組込み機器 6 製品（BD-CAM，アンドロイド携帯電話，液晶プロジェクタなど）24 件の障害の解析困難な障害（5 営業日以内に担当者が解決出来なかった障害）に適用し，そのうち 18 件で効果を確認した．そこで，本障害解析環境がどのような障害に有効であったかを分析した．その結果を以下に示す．

まず．発生した障害の顕在化箇所を表 6 を用いて説明する．それらの障害の顕在化箇所は，テスト時においてほぼアプリの障害として検出されたものであった．

表 6　障害顕在化箇所

#	障害顕在化箇所	件数	比率
1	アプリ	17	94.44%
2	他	1	5.56%

次に実際に障害の原因となった箇所を表 7 を用いて説明する．その多くは#1,2 にある様に OS，ドライバといったプラットフォーム部分で起きた障害であった．また，#3 アプリにおいても，障害が顕在化したアプリ以外の，別のアプリが原因であった．

表 7　障害原因箇所

#	障害原因箇所	件数	比率
1	OS	5	27.78%
2	ドライバ	7	38.88%
3	アプリ	3	16.66%
4	ミドル	1	5.56%
5	ツールチェーン	1	5.56%
6	データ	1	5.56%

以上の結果より，本障害解析環境は，障害発生個所と顕在化箇所が別にあるような障害に対して有効であることが明らかになった．また，それらの多くは OS やドライバ，ミドルといったアプリ以外の導入品に含まれる障害であった．本障害解析環境が有効でなかったケースは，各機能固有の解析ツールや，メーリングリストの情報などで解決した事例であり，本環境を使う前の調査段階で対応が可能であった．

6. おわりに

本研究では，組込みシステムにおける障害解析環境について検討し，Armadillo を用いた試験環境及び，日立家電製品を中心とした製品開発において評価を行った．課題抽出にあたっては，それまで主に DTV 及びビデオカメラに代表される家電製品の開発において，開発工数に悪影響を与えていた作業を分析し，対策が必要な課題 3 点と，その課題に着目した要件 3 点を抽出した．それをもとにそれらを解決するための①解析対象絞り込み機能，②長時間トレース機能，③形式変換機能からなる構成で障害解析の効率改善機能を設計し，Armadillo を用いた試験環境と，DTVHR-01 で評価し，その後組込み製品 6 製品の実開発に適用し，その有効性を確認出来た．

現在，これらの環境は当時の環境から一部変更し活用している．OS トレーサは LKST から現在の Linux トレースの標準となっている Ftrace に移行した．また OProfile も Perf に移行し，日立グループ内の産業機器などの開発で適用を続けている．

参考文献

[1] 田中勇樹, 石郷岡祐他： 高応答性と統合容易性を両立するマルチコア活用ソフトウェア統合ミドルウェア, 情報処理学会研究報告, Vol.2019-EMB-50 No.6(2019)

[2] 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,pp.930-938(2001)

[3] Jonathan B. Rosenberg 著，吉川邦夫訳: デバッガの理論と実装 アスキー出版局（1998）

[4] 畑崎恵介, 中村哲人, 芹沢一:システムの挙動に対応して動作の切替えが可能なイベントトレーサ LKST の開発, 情報科学技術フォーラム一般講演論文集 2002(1), pp.177-178(2002)

[5] Patricia, C and Aurelie, B et al. Debugging embeded multimedia application traces through periodic pattern mining, Proc. 10th ACM international conference on Embeded Software(EMSOFT'12), pp 13-22(2012).

[6] Paul, G. and Bharat, J.: Methodology and archtecture of JIVE, Proc. ACM symposium on Software visualization(SoftVis '05), pp95-104(2005)

[7] 後藤 隼弐,本田 晋也,長尾 卓哉,高田 広章,トレースログ可視化ツール TraceLogVisualizer (TLV),コンピュータソフトウェア,2010,27 巻,4 号,p. 4_8-4_23(2010)

[8] David Legendre, Francois Audeon: Detection and Resolution of Real-Time Issues using TimeDoctor https://elinux.org/images/3/3e/Detection-of-RT-issues-with-TimeDoctor.pdf (2020 年 5 月 13 日現在)

[9] Steven rostedt ftrace – Function Tracer , https://www.kernel.org/doc/Documentation/trace/ftrace.txt (2020 年 5 月 13 日現在)

[10] Mohamad Gebai, Michel R. Dagenais: Survey and analysis of kernel and userspace tracers on Linux: design, implementation, and overhead, ACM Computing Surveys, vol. 51, no 2(2018)

[11] Kernel Probes(Kprobes), https://www.kernel.org/doc/Documentation/kprobes.txt (2021 年 6 月 25 日現在)

[12] LTTng is an open source tracing framework for Linux, https://lttng.org/ (2021 年 5 月 18 日現在)

[13] Linux kernel profiling with perf, https://perf.wiki.kernel.org/index.php/Tutorial(2021 年 7 月 29 日現在)

[14] Oprofile, https://oprofile.sourceforge.io/news/(2021 年 7 月 29 日現在)

[15] 長野 岳彦, 亀山 達也.: 組込みトレース技術とその応用, 研究報告組込みシステム（EMB）, 2011-EMB-20, PP1-6(2011)

[16] 長野岳彦: OProfile porting on MIPS architecture, CE Linux Forum Japam technical jamboree #16 講演資料 (2007)