情報処理学会論文誌 Vol.58 No.8 1336–1347 (Aug. 2017)

圧縮機能を備えた組込みシステム向き分離型Linuxプロセストレース機構

プラウィーン アモーンタマウット[1,a)] 早川 栄一[2,b)]

受付日 2016年11月18日, 採録日 2017年5月16日

概要：組込みシステムを対象とした省メモリのLinuxプロセスのトレース機構の開発と評価を行った．組込みシステムでシステム動作のトレースを行う場合，トレースデータを保存するメモリ容量に制限があるため，プロセスのコンテキストスイッチのトレースデータが消失する可能性がある．この問題を解決するために，Linuxカーネル2.6.29が提供しているトレース機構であるFtraceに対して，低オーバヘッドのトレースデータ圧縮機構を追加した．さらに，カーネルプロセスレベルでトレースデータを外部に転送する機構を追加した．これによって，トレース対象およびトレースデータを扱うユーザプロセスへのプロセススケジューリングへの影響を低減しつつ，システム内のメモリを圧迫しないイーサネット経由の分離型Linuxプロセスのトレース機構を実現した．

キーワード：組込みシステム，Linux，プロセストレース機構，Ftrace

A Separated Linux Process Tracing Mechanism with Compression for Embedded Systems

Praween Amontamavut[1,a)] Eiichi Hayakawa[2,b)]

Received: November 18, 2016, Accepted: May 16, 2017

Abstract: A process tracing mechanism for Linux that can operate on less memory embedded system is developed and evaluated. When acquiring context switch log from the logging system, some part of log data may be lost because limited buffer size causes log data to overflow. Exporting log data to out of the kernel method affects user process scheduling. In order to solve this problem the original trace mechanism of Linux kernel 2.6.29 is enhanced to a new trace mechanism with context switch logging function that is available on less memory embedded system. Less overhead compressing function into the trace mechanism and directly exported to an external system in kernel level are added.

Keywords: embedded system, Linux, process Tracing, ftrace

1. まえがき

システムの高機能化やネットワークにともない，ネットワークからデバイス操作やデータ転送を行うために，Linuxをベースに組込みシステムを構築することが増加している．Linuxが持つ移植性の高さやネットワークの安定性などから，Android [1] による携帯電話・スマートフォンや，TV，ホームサーバからロボットまで多くのデバイスで用いられつつある．このような組込みシステムでは，デバイス制御やGUI部分でプロセスやスレッドを用いて実装することが多い．システムの応答性の向上や，制御プログラムの動作を把握するには，プロセスやスレッドに関して実行時のプロセス情報を取得することが重要になる．特に，スマートフォンの評価ボードKZM-A9におけるAndroidのGUIの応答性向上のため，OSのプロセスを可視化することで，

[1] 拓殖大学大学院工学研究科電子情報工学専攻
Takushoku University, Graduate School of Engineering, Electronics and Information Sciences, Hachioji, Tokyo 193–0985, Japan
[2] 拓殖大学工学部情報工学科
Takushoku University, Faculty of Engineering, Department of Computer Science, Hachioji, Tokyo 193–0985, Japan
a) praween@hykwlab.org
b) hayakawa@cs.takushoku-u.ac.jp

プロセスの状態の行動を分かりやすくするウェブブラウザ上の可視化ツールが開発された [2]．Android は Linux をベースとしたプラットフォームであるので，Android のプロセスを理解するには Linux のプロセスやスレッドをトレースすることになる．

Linux では，複数のプロセスやスレッドのトレースは，デバッグファイルシステムという疑似ファイルを介してカーネル空間にあるトレースデータを読み書きすることで，プロセスのコンテキストスイッチやイベントのプローブ，関数単位での実行詳細を，トレースデータとして取得することができる．これらのデータは人間が読みやすいように，ASCII 文字列で取得することができる．ユーザ空間では，これらのトレースデータを取得し，格納することで，カーネルの起動から任意の時刻までのシステム動作の監視が可能になる．これまで，トレースデータを効率良く扱い，分かりやすく表示するためのツール群が開発されてきた [3], [4], [5], [6], [7].

しかし，上記のトレースメカニズムを，組込みシステムに適用することには問題がある．

第 1 に，利用者の行動やイベントによって，保存すべきトレースデータのサイズが変化し予測が難しい点である．組込みシステムでは長期にわたって稼働することが多く，トレースデータのためのメモリを事前に多く確保しておく必要がある．しかし，一般的に組込みシステムに搭載されているメモリは少なく，大きくなりがちなトレースデータをシステム内に保持することが難しい．

第 2 に，コンテキストスイッチ情報を取得，保存するときに，その機構自体がアプリケーションプロセスの動作に影響を与えるという問題がある．たとえば，起動するプロセスの数が多い場合，一般的に生成されるトレースデータのサイズは増加する．カーネル内部でトレースデータを保持するメモリサイズには制約があることから，データ消失を防ぐにはユーザ空間への頻繁なデータのコピーが必要になる．このコピー自体がコンテキストスイッチを生じさせて，測定対象となるアプリケーションの動作に影響を与えてしまう．コンテキストスイッチ数やトレースデータ保持のメモリサイズを低減させようとしてコピーの回数を減らした場合，カーネル内部でトレースデータが消失してしまう可能性がある．これは，Linux カーネルではコンテキストスイッチ情報の取得にはリングバッファを用いているので，リングバッファからコピーできなかった過去のトレースデータは上書きされてしまうからである．

これらの問題を解決するために，本論文では組込み Linux をターゲットとして，プロセスやスレッドのプロセス生成およびコンテキストスイッチ時のトレースデータのサイズ低減手法を提案する．組込みシステム向けの低オーバヘッドなコンテキストスイッチに関するトレースデータ圧縮機能を実装し，その有効性を評価した．さらに，プロセススケジューリングへの影響を抑えつつ，トレースデータ保存のメモリ使用量を減らすために，ネットワーク経由でトレースデータを取得可能な分離型 Linux プロセストレース機構を実現した．この 2 つの機構を用いることで，ユーザ空間を経由したリモートマシンへのトレースデータコピーと比較して，従来の Ftrace を用いてローカルディスクへ保存した場合とほぼ同等のオーバヘッドで，リモートマシンへトレースデータを転送することが可能になった．また，アプリケーションの動作への影響や内部のメモリへのスイッチへの影響を抑えたことから，長期にわたって稼働するネットワーク接続された組込みシステムへ適用することが可能になった．

2 章では従来のトレース機構について述べ，従来機構が抱える問題を分析する．3 章ではこの問題を解決するトレース機構の設計について述べる．システムの全体の機能および全体構成をついて述べ，トレース機構の中核部分であるプロセストレースデータの特性に応じた圧縮機能の設計および解凍方法を述べる．4 章では，従来のトレース機構およびユーザ空間を介したトレースデータの転送方法と比較した，本機構の評価と考察を行う．5 章では関連研究について述べ，6 章で成果についてまとめる．

2. 従来のトレース機構

本章は，従来の Linux のトレース機構について述べる．

2.1 Ftrace：Linux のトレース機構

本システムは，Linux カーネル 2.6.29 に含まれる軽量なカーネルトレースである Ftrace [8], [9] をベースとして開発した．Ftrace は Linux カーネル内部付属のトレーサであり，2.6.27 から追加された．ユーザ空間のトレースツールは，この機構を通してカーネルの様々な動作結果を容易に取得できる．Ftrace はカーネルバージョンによって提供される機能が異なるが，本システムで対象とする Ftrace の機能は次のとおりである．

- function：すべてのカーネル関数をトレースする．
- function_graph：すべてのプローブ可能なカーネルおよびユーザプロセス内に実行する関数をトレースし，呼び出しグラフ化した文字列を出力する．
- sched_switch：カーネルのスケジューリングにより発生したプロセス状態の遷移をトレースする．

本システムでは，プロセス切替えに関する情報を出力する sched_switch に注目した．本章では，この部分の機構の概要について述べる．

2.2 トレース機構の動作

従来のトレース機構を**図 1** に示す．トレース機構は，トレース内部機構（Trace internal mechanism）とトレース外部機構（Trace external mechanism）の 2 つから構成さ

れている．

トレース内部機構は，プロセススケジューリングを管理するプロセス管理サブシステム（Process subsystem）と関連して，コンテキストスイッチ時のトレース情報を生成する機能を提供する．これをトレースサブシステム（Trace subsystem）と呼ぶ．トレースサブシステムは，トレース外部機構のトレース設定プログラム（Trace setting program）で設定されたパラメータに従ってトレースデータを取得する．たとえば，デバッグファイルシステムの trace_enable を 1 に設定すると，リングバッファは初期化されてトレースデータを収集するようになる．プロセスに関するトレースデータを取得するときにプロセスをブロックすることはなく，設定したリングバッファのサイズを超えた場合は過去のデータは上書きされる．

トレース外部機構は，カーネル空間内にあるトレース内部機構から，ユーザ空間上にあるユーザプロセス（Application programs）に，I/O サブシステム（I/O subsystem）を通じてトレースデータをエクスポートする．このとき，トレース内部機構は，トレースポイントにフラグを付けたデータに対して，リングバッファからエクスポートバッファにトレースデータを収集する．これを文字列化機構（String Converter）によって ASCII 文字列に変換した後にデバッグファイルシステムの tracepipe を通じてユーザ空間にあるトレースデータ取得プログラム（Log acquiring program）にエクスポートする．

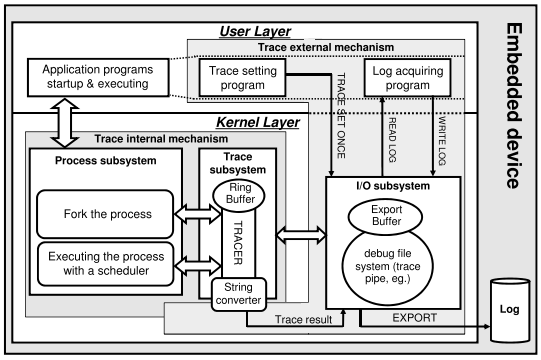

図 1 従来のトレース機構

Fig. 1 Exsisting Linux (v2.6.29) tracing mechanism.

2.3 コンテキストスイッチのトレースデータ

図 2 に Ftrace が生成するコンテキストスイッチのトレースデータ（以下，Ftrace 形式）の実例を示す．このデータは，プロセス生成やプロセス状態の切換え時のプロセス情報を Ftrace を用いてトレースしたときの情報である．

Ftrace 形式は，リングバッファに格納されるバイナリデータを，プログラムで扱いやすくするために，図 2 のように ASCII 文字列化しエクスポートバッファに収集した後，ユーザ空間で動作するプロセスにエクスポートする形になっている．

2.4 従来のトレース機構の問題点

従来の Ftrace 形式を組込みシステムで用いる場合の問題点は次の 3 点である．

(1) リングバッファがトレースデータ生成とユーザ空間へのコピーとの速度差を吸収できず，データが消失する可能性がある．Linux カーネルでは，エクスポートバッファのサイズは 1 ページに限定されている．コンテキストスイッチのトレースデータは ASCII 文字列に変換し，ユーザ空間へエクスポートされるので，データのエクスポートが間に合わない可能性がある．この場合，リングバッファ上のデータは上書きされ，データが消失してしまう．

(2) トレースデータのユーザ空間へのコピー時に発生したコンテキストスイッチが，プロセススケジューリングに影響を与える可能性がある．ユーザプロセスは，トレースデータをデバッグファイルシステム経由でユーザ空間へコピーする．データコピー自体はスケジューリングの対象になることから，トレースデータの取得や保存がスケジューリングそのものに影響を与えてしまう．スケジューリング自体への影響を避けるには，コピーするプロセスの優先度を下げるなどの方法があるが，この場合データがカーネル内部に保持されることから，(1) で述べたデータ消失の危険が生じる．

The meaning of each data in data constructor												
Process name	Entry process ID	CPU core number	Timestamps (seconds)	Timestamps (microseconds)	Previous operation of process ID	Previous operation of process priority	Previous operation of process state	Operational entry type	Next operation of CPU core number	Next operation of process ID	Next operation of process priority	Next operation of process state
{comm}	{entry_pid}	{cpu}	{secs}	{usec_rem}	{prev_pid}	{prev_prio}	{prev_state}	{entry_type}	{next_cpu}	{next_pid}	{next_prio}	{next_state}

String convertor

{DATA NAME}	{comm}-{entry_pid} [{cpu}] {secs}.{usec_rem} {prev_pid}:{prev_prio}:{prev_state} {entry_type} [{next_cpu}] {next_pid}:{next_prio}:{next_state}
Context switch	cat-42 [000] 4154504421.591616: 42:120:S + [000] 42:120:S cat-42 [000] 4154504421.591616: 42:120:S ==> [000] 0:140:R

図 2 コンテキストスイッチトレースデータの例

Fig. 2 Example of a context switch trace data.

(3) トレースデータの保存容量に限界がある．多くの組込みシステムは機器内のフラッシュ ROM の容量が小さいことから，機器内に大量のデータを保存することはできない．たとえば，カーネル 2.6.29 のデフォルト状態の Android Froyo でブラウザを起動させ，タッチパネルをタッチし続けると，毎秒 135 KBytes 程度のトレースデータを生成する．1 日あたり約 11.7 GB の大きさとなってしまうことから，長時間稼働するシステムのトレースデータを組込み機器の内部に保持するのは難しい．

3. 組込み Linux 向け分離型トレース機構

3.1 設計方針

前章で述べた問題を解決するために，ネットワーク接続された組込み Linux を対象とした，分離型トレース機構を設計した．従来の Ftrace に対して，プロセスの状態遷移に関するトレースデータの取得オーバヘッドを低減する改良を行った．また，組込みデバイス側への保存サイズを減らすために，トレースデータを保存する部分を “Log Acquiring Server” として組込みシステムの外部に分離した．本機構の設計方針は次のとおりである．

(1) **カーネル内部のトレースデータを圧縮する**：カーネル内部で多くのデータを保持可能にするために，カーネル内部でのトレースデータに対してデータ構造に依存した圧縮を行い，より多くのデータを保持できるようにする．

(2) **トレースデータを生成する計算機とデータを保存する計算機を分離する**：組込みシステムの主記憶および二次記憶のサイズは制限されていることから，トレースデータをすべて組込み機器内部で保持するのは難しい．また，ネットワーク接続された組込みシステム環境では，複数の組込みシステムからトレースデータを収集する機構が必要になる．そこで，トレース対象の計算機にデータを保存せず，ネットワークを介して外部の計算機へのコピーを行う．

(3) **ユーザ空間へのデータコピーを極力さける**：スケジューリングへの影響を避けるために，ユーザ空間へのトレースデータのコピーは行わず，カーネル内部だけで処理する．

3.2 トレース機構の設計

本システムの全体構成を**図 3** に示す．本システムでは，トレースデータ生成側（Log Generation Part）の組込み機器（以下生成側）とトレースデータ取得側（Log Collection Part）のログ収集計算機（以下取得側）の 2 つから構成する．これによって，トレースデータの保存は生成側の組込み機器から分離されるので，連続したサイズの大きいトレースデータの保存にかかるオーバヘッドを減らすことができる．

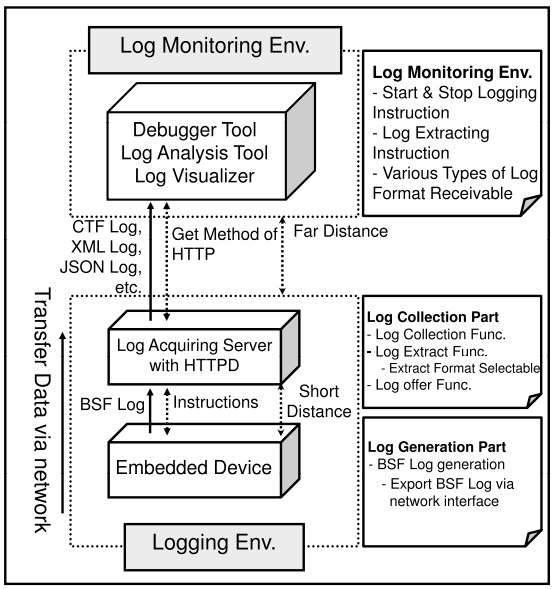

図 3 システムの全体設計
Fig. 3 Overall structure of the system.

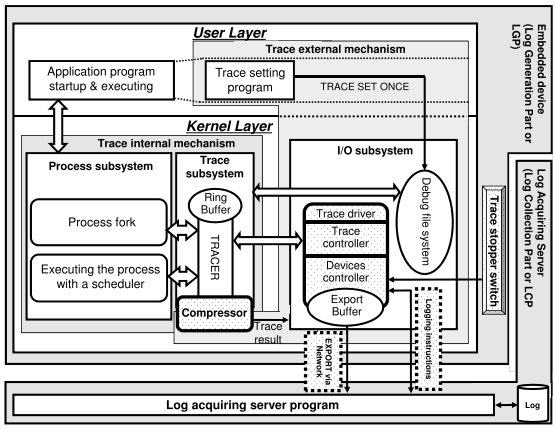

図 4 提案したトレース機構の全体構成
Fig. 4 Structure of Proposed tracing mechanism.

本システムでは従来のカーネルスケジューラとそれに関連したコード，およびトレースデータを生成しエクスポートする機構は変更せず，そのまま利用している．これは，トレース内部機構がプロセス管理部との関連性が高く，この部分の変更はカーネル全体に影響を与えるからである．カーネルスケジューラとトレース内部機構の構造体を変更することは，カーネルコードの変更に対する追従性を悪化させることから，本システムでは採用していない．

本トレース機構の構成を**図 4** に示す．図 4 の生成側に，従来の機構に圧縮機能（Compressor）を持ったトレースドライバ（Trace Driver）を追加する．トレースドライバはカーネル空間（Kernel Layer）で動作するサブシステムであり，次の 2 つの役割を備えている．

- **トレース制御部（Trace Controller）**は，トレースサブシステム（Trace Subsystem）の確認機構とデータ

圧縮機能を備える．前者はシステムの起動時にトレースの有無を確認する．後者はトレーサが有効の場合にだけ起動する．圧縮機能の設計は次の節で述べる．

- **デバイス制御部（Device Controller）**は，トレースデータ取得サーバプログラム（Log acquiring server program）にデータを転送する．このとき，トレースデータをユーザ空間へコピーせず，直接トレースデータ取得サーバにTCP通信によってエクスポートする．また，帯域の狭い通信メディアの場合でも，リングバッファに保持するデータおよびネットワークで転送するデータを圧縮することで，従来の方法でのデータ転送と比較してシステムのメモリ消費を抑えつつ，データを転送することが可能になる．
  エクスポートの処理は従来のトレースと同様に100 m秒ごとのタイマ割込みで行われる．そして，“Logging Instructions”は組込み機器に対しトレースデータの取得の開始停止の命令と定義する．本命令は，1) トレースの開始，2) トレースの停止の2種類の命令がある．これらの命令は，トレースデータ取得サーバからトレース制御部を送ることで，外部からトレースの開始，停止の指定が可能になる．

トレースデータ取得サーバは，トレースの開始命令・トレースの停止命令を組込システム側に渡すことによって，組込みシステム側からのすべてのトレースデータを取得して保存し，データ解凍機能により圧縮されたトレースデータを解凍する．取得側への転送は図1のようにユーザ空間へのコピーをせず，上記のトレースドライバによってカーネル内部から直接行う．

3.3 圧縮手法の設計

圧縮機能（Compressor）は，トレースドライバのトレース制御の機能として，従来の文字列化機能に追加することで実現する．プロセスへの影響を減らし，より長い時間トレースデータ保存するには，従来の文字列化機能より圧縮率が良く，低オーバヘッドな方法を検討する必要がある．従来の文字列化機能でのデータ構造を**表1**に示す．これに対して，本圧縮機能でのデータ構造を**表2**に示す．従来のFtrace形式では，リングバッファから出力されたデータを元データよりサイズが大きい文字列に変換して，サイズが小さいエクスポートバッファに格納している．これに対して，本圧縮機能では文字列化機能と同様にデータをリングバッファからエクスポートバッファに格納するが，データは特徴に応じて文字列化せず，よりデータサイズの小さいデータ表現を用いてエクスポートバッファに格納するようにする．

このデータ構造の特徴を活かした形で圧縮するために，従来のFtrace形式の特徴を分析したところ，特徴は次の3点であることが明らかになった．

表1 文字列化機能でバッファに格納する領域の比較
Table 1 The comparison of buffering regions in string converter.

Data name	Data size in ring buffer (Bytes)	Data size in export buffer (Bytes)
comm	16	16
entry_pid	4	7
cpu	4	5
secs	8	2∼11
usec_rem	8	7
prev_pid	4	7
prev_prio	1	4
prev_state	1	3
entry_type	1	3
next_cpu	4	5
next_pid	4	7
next_prio	1	4
next_state	1	3
Total	57 Bytes	Min 74∼Max 83 Bytes

表2 データの特徴による分類と圧縮
Table 2 Classification and compression by data characteristics.

Data name	Data charecter-istic	Data size in ring buffer (Bytes)	Data size in export buffer	
			Binary section (bits)	String section (Bytes)
comm	(A)	16	5	0∼16
entry_pid	(C)	4	32	null
cpu	(A)	4	2	0∼3
secs	(A)	8	5	0∼10
usec_rem	(A)	8	5	0∼6
prev_pid	(C)	4	32	null
prev_prio	(C)	1	8	null
prev_state	(B)	1	3	null
entry_type	(B)	1	2	null
next_cpu	(A)	4	2	0∼3
next_pid	(C)	4	32	null
next_prio	(C)	1	1	null
next_state	(B)	1	3	null
Total		57 Bytes	17 Bytes	min 0 ∼ max 38 Bytes
			Min 17 ∼ Max 56 Bytes	

(A) データの表現がメモリの確保領域より広く，同じデータの繰返し頻度が高いデータ
(B) データの表現がメモリの確保領域より狭いデータ
(C) データの表現がメモリの確保領域において十分なデータ

(A) に属するデータは，プロセス名や時間などを表現する“comm，cpu，secs，usec_rem，next_cpu”である．こ

れらは可変長文字列なので，文字列表現である文字列部（String section）とこれにおけるデータ長を表すバイナリ部（Binary section）から構成する．特に，バイナリ部は文字列部の最大文字数と相当するビットのデータ表現の領域を確保するようにする．そして，事前に出力したデータと一致したデータならば文字列は出力せず，文字数を表すバイナリ部は 0 のデータを書き込む．comm を例として，データが “er.ServerThread” で 15 文字の文字列長の場合を考える．ここで，表 1 のエクスポートバッファを参照すると，comm はリングバッファとエクスポートバッファの格納領域が 16 バイトである．これで，表 2 のように文字列部の確保領域の最小値・最大値を 0〜16 バイトにするように定義することができる．バイナリ部は 16 バイトを表す 5 ビットのメモリ領域を用意にすればよい．このとき，文字列部に文字列表現の “er.ServerThread” を書き込み，バイナリ部に 15 を書き込む．これは 1 つ前の出力データと見なして，次の出力データを考える．出力データが 1 つ前の出力データと同じ “er.ServerThread” となる場合は，comm の文字列部に何も書き込まないようにする．これで，comm の文字列部は 0 バイトになるため，バイナリ部には 0 を書き込む．

(A) のデータは繰返し頻度が高いので，これによって圧縮率を向上させることができる．しかし，文字コードに変換や文字列の長さを計算するオーバヘッドが存在する．このオーバヘッドを吸収するために (B) と (C) に属するデータを次に考える．

(B) に属するデータは，プロセスの状態を表す “prev_state, entry_type, next_state” である．これらのデータの特徴は，いずれも少ない固定のビット数で表すことができる点である．そこで，ビットフィールドによりビット表現を縮め，バイナリデータをバイナリ部にだけに書き込むようにする．たとえば，prev_state のデータを考える場合，これは前のプロセス状態を表すデータであり，8 つの状態だけで表すことができる．表 1 のリングバッファの格納領域を参照すると，文字化した 1 バイトデータを固定 3 バイトの文字列のデータにする必要はないので，表 2 のように 3 ビットのバイナリデータにすれば十分である．(B) のデータは文字列で表現する必要はないことから，データのサイズは小さくなり，転送オーバヘッドが (A) より低くなる．

(C) に属するデータは，プロセス ID や優先度などを表す “entry_pid，prev_pid，prev_prio，next_pid，next_prio” である．これらのデータはシステムの起動中に生成する動的なデータなので，データの確保領域は予測できない．さらに，表 1 のリングバッファ格納領域を参照すると，リングバッファに格納する領域はエクスポートバッファよりも少ないので，格納領域を縮める必要がないデータである．このことから表 2 では，そのままバイナリコードでバイナリ部だけにデータを書き込むようにする．データ自体を加工せずに書き込むことで，オーバヘッドを低くすることができ，(A) のオーバヘッドを吸収することもできる．

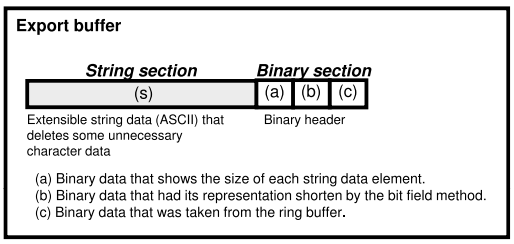

図 5 圧縮機能によりエクスポートバッファに格納する 1 行のトレースデータ

Fig. 5 One line of tracing data stored in export buffer with compression.

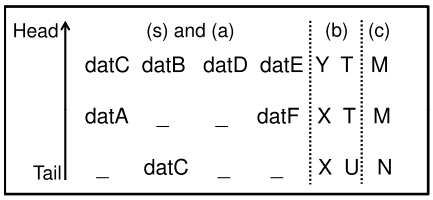

図 6 中間ログのデータ構造

Fig. 6 Data structure of intermediate log.

これらの特徴に基づいて，データの特徴で分類し，**図 5** のようにエクスポートバッファに書き込む．可変長の文字列としては “comm，cpu，secs，usec_rem，next_cpu” の文字列部 (s) をこの順番に書き込む．

バイナリ部は，バイナリデータの情報自体と文字列部の各要素の文字列長のデータを持つバイナリヘッダも格納する．バイナリヘッダは次の 3 つのデータ (a)〜(c) を書き込む．(a) は “comm，cpu，secs，usec_rem，next_cpu” のデータ，(b) は “entry_type，prev_state，next_state” のデータ，(c) は “entry_pid，prev_pid，next_pid，prev_prio，next_prio” のデータを，それぞれこの順序で書き込む．

また，バイナリヘッダは後方に配置する．この理由は，(a) の可変長文字列の各要素の文字列長をまとめるのに発生するオーバヘッドを減らすためである．

このフォーマットを The mixed Binary and String Format（以下 BSF）と呼ぶ．これは文字コードとバイナリーコードを連続した形で構成しているからである．このフォーマットを使用するトレースデータを BSF Log と呼ぶ．

3.4 データの解凍方法

末尾から固定サイズのヘッダバイナリを読み込み，図 5 における (a)〜(c) の各データを**図 6** のような中間ログとして解析する．(a) のデータ部分は各要素の文字列の長さをバイナリ部から読み込み，各要素の文字列を分離する．文字列の長さが 0 の場合は，文字列部にデータがエクスポートされていない．これによって要素の分離が困難になるこ

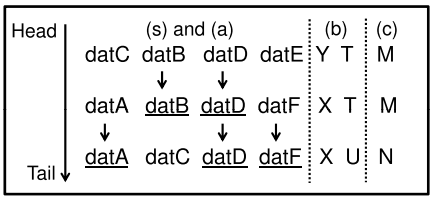

図 7 解凍した最終ログのイメージ
Fig. 7 Image of extracted log.

とを回避するため，マーカを “ _ ” の 1 バイトに格納する．そうではない場合は，その要素の長さだけを各要素の文字列に格納する．(b) と (c) のデータ部分はそのまま文字列化し，格納する．

解凍自体のオーバヘッドによる組込みシステムの影響を避けるために，解凍機能は，“Log Acquiring Server” で提供する．上記に述べたデータの解凍方法で BSF ログを解析した後，**図 7** のように先頭から読み込み，解凍することで従来の Ftrace 形式，JSON 形式および XML 形式に変換する．

4. 圧縮機能付きトレースの評価

4.1 評価システムの環境

本提案手法を，京都マイクロコンピュータ株式会社の KZM-A9 評価ボード上に動作する Linux カーネル 2.6.29 およびそのうえで動作する Android 2.2 をベースにシステムプロトタイプを実装し評価した．**表 3** は評価環境を示す．

本システムを実装するために変更した部分を，**表 4** に示す．本提案トレースを動かすために，該当カーネルのコードに 17 行のパッチが必要である．また，カーネル内からソケット通信を行うために ksocket モジュール [11] を用いた．これは，カーネル内でソケット API を呼び出すためのライブラリである．

4.2 評価実験と結果

本評価実験では，ユーザ空間でトレースデータ取得プログラムを起動させ，Ftrace を用いてネットワークを経由せずに SD カードに保存するトレース機構（以下，“Existing Trace”）とトレースデータをネットワーク経由で転送した場合（以下，“Existing Trace via Ethernet”）を基準にして，提案トレース機構（以下，Proposed Trace）を評価する．最後に，“Existing Trace with Existing Compressor（gzip）” と “Existing Trace via Ethernet with Existing Compressor（gzip）” はユーザ空間従来のトレース機構を用いて，既存圧縮手法である gzip と比較し本提案手法の妥当性を評価する．

“Existing Trace” と”Existing Trace via Ethernet” は，Ftrace の sched_switch で実験を行った．なお，本評価実験ではネットワークに 100 Mbps のイーサネットを用い，

表 3 ハードウェアの仕様 [10]
Table 3 Specification of Hardware for System Evaluation Experiment.

KZM-A9 評価ボード	
CPU	ARM Cortex-A9MPCoreTM Dual CPU
	Operating Frequency 533 MHz
Memory	Main Memory 512 MB（DDR2-533）
	Internal SRAM 128 KB
Touch Panel	Controller: EMMA-EV2 on-chip display
	LCD: WVGA（800*480 Panel, RGB（8:8:8））
	HDMI: Sil9024A（HDMI1.3）

表 4 カーネルコードの変更
Table 4 Changed list of Linux kernel code.

Kind	File Name	Patch (Lines)	Add (Lines)
Kernel internal code	trace.c	16	0
	trace.h	1	0
Trace driver code	trace_driver.c	0	2,523
	trace_getmajorid.c	0	85
	compresslog.h	0	33

cpufreq の機能を用いず CPU 周波数は 533 MHz 固定とした．

タイマ割込みについては NO_HZ 指定による変動周期タイマを用いた．変動周期タイマでは，タイマ割り込みは必要なときだけに発生する．また，実験に用いた Linux では，高精度なハードウェアタイマは使用していない．

本実験では，ユーザがシステムに対して一定の行動を行うようにして，実験環境の再現性を確保した．具体的には，Android のブラウザ[*1]を起動させて，重さ 500 g のテキストブックをタッチパネル上に載せた．評価ターゲットのマシンには抵抗膜式のタッチパネルが搭載されているので，これによってパネルから同じデータを取得することが可能になる．実験はこの時点からトレースの実行時間（T）を 10 秒ごとに増やし，150 秒までの 15 回のデータを 1 セットとして，3 セット試行を行い，その平均値を求めた．

評価実験は次の評価項目に着目した．

1) **エクスポートされたトレースデータのサイズ（Exported Log Data Size）**
各トレース機構において，エクスポートされたトレースデータのサイズを表す．
- **Existing Trace** では，トレースデータを圧縮せずに，SD カードにおける入出力を用いた場合のサイズを表す．本トレースは従来の Ftrace 形式でエクスポートする．
- **Existing Trace via Ethernet** では，データを圧縮

[*1] User-Agent: Mozilla/5.0 (Linux; U; Android2.2; ld-us; kmc_kzm9d Build/FRF91), AppleWebKit533.1 (KHTML, likeGecko), Version/4.0 Mobile Safari/533.1

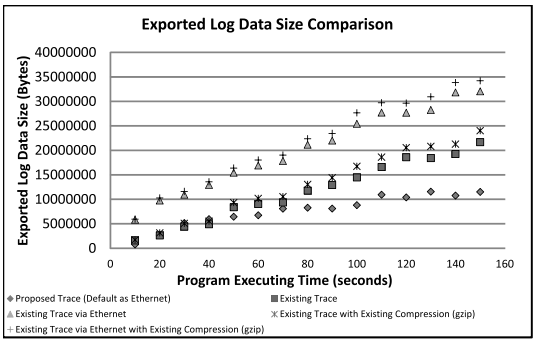

図 8 データのサイズの比較
Fig. 8 Comparison of data size.

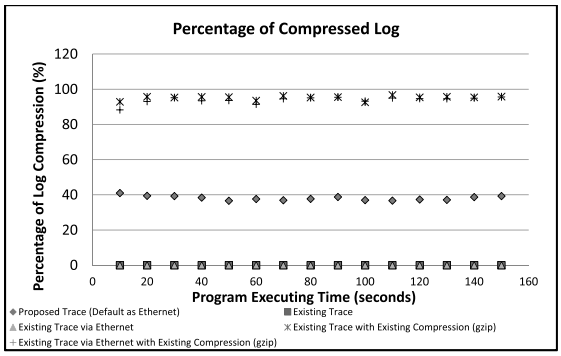

図 9 コンテキストスイッチ数の比較
Fig. 9 Comparison of the number of context switches.

せずに，ユーザ空間でイーサネットを経由したデータ転送を行った場合のサイズを表す．本トレースは従来の Ftrace 形式でエクスポートされる．

- **Existing Trace with Existing Compressor**（**gzip**）では，Existing Trace と同様に従来の Ftrace 形式でエクスポートするトレースのデータサイズとこのデータを gzip で圧縮するデータのサイズを表す．
- **Existing Trace via Ethernet with Existing Compressor**（**gzip**）では，Existing Trace via Ethernet と同様にトレースデータのサイズとこのデータをユーザ空間に gzip で圧縮した後に，ユーザ空間でイーサネットを経由したデータ転送を行った場合のサイズを表す．本トレースは従来の Ftrace 形式でエクスポートされる．
- **Proposed Trace** では，データをカーネル空間に圧縮し，カーネル空間でイーサネットを経由したデータ転送を行った場合のサイズを表す．本トレースは BSF ログ形式でエクスポートする．

実験結果を**図 8** に示す．すべての方式において時間とともにトレースデータサイズは増加しているが，“Proposed Trace” は “Existing Trace via Ethernet” と比較してデータサイズの増大がゆるやかであり，SD カードを用いた既存のトレース方式とほとんど同じデータサイズで転送することが可能である．

2) **コンテキストスイッチ数**（**The Number of Context switch**）は，プロセスのディスパッチ時にコンテキストを切り替えたプロセスの回数を表す．

- **Existing Trace** では，トレースデータを圧縮せずに SD カードへ入出力を行った場合の回数を表す．
- **Existing Trace via Ethernet** では，トレースデータを圧縮せずに，ユーザ空間へエクスポートし，イーサネットを経由したデータ転送を行った場合の回数を表す．
- **Existing Trace with Existing Compressor**（**gzip**）では，トレースデータを SD カードに保存した後に gzip で圧縮した後の場合の回数を表す．
- **Existing Trace via Ethernet with Existing Compressor**（**gzip**）では，トレースデータをエクスポートした後にユーザ空間で gzip で圧縮した後に，イーサネットを経由したデータ転送を行った回数を表す．
- **Proposed Trace** では，トレースデータを圧縮し，カーネル空間でイーサネットを経由したデータ転送を行った場合の回数を表す．

実験結果を，**図 9** に示す．いずれの方式も時間の経過とともにコンテキストスイッチ数は増加している．ユーザ空間からイーサネット経由でデータを転送する “Existing Trace via Ethernet” が一番多く，SD カードを用いた “Existing Trace” が最も少ない．“Proposed Trace” は一部で “Existing Trace” よりもコンテキストスイッチ数が増えるものと，“Existing Trace via Ethernet” よりも少ない数となっている．

3) **割込みの頻度**（**Interrupt Frequency**）は，上記のトレースを行う最中に発生した割込み頻度を表す．

- **system_timer** は，CPU0 へのタイマ割込みの頻度である．
- **system_timer1** は，CPU1 へのタイマ割込みの頻度である．
- **emxx_sdc** は，SD カードにおける入出力の割込みの頻度である．
- **eth0** は，イーサネットでのネットワークのデータ転送による割込みの頻度である．
- **kzm9d_touch** は，タッチパネルに対する入出力の割込みの頻度である．

実験結果は**図 10** に示す．提案手法は，“Existing Trace” よりも割込み頻度は高いが，“Existing Trace via Ethernet” と比較して，割込み頻度を低減していることが分かる．

4) **ログの圧縮率**（**Percentage of Compressed Log**）
本提案トレース機構の圧縮機能によるデータの圧縮率を表す．圧縮率は，本システムの “Log Acquiring Server” において BSF ログ形式を Ftrace 形式のトレー

表 5 130 秒実行した場合のシステムのオーバヘッド
Table 5 Overhead of system for executing 130 seconds.

	Existing trace	Existing trace via ethernet	Existing trace with existing compressor (gzip)	Existing trace via ethernet with existing compressor (gzip)	Proposed trace
Data Size (Byte)	18,902,754	17,261,556	922,059	1,678,149	10,380,256
Compressor (sec)			25 (19.23%)	33 (25.38%)	0.948 (0.73%)
String converter (sec)	1.15 (0.88%)	0.926 (0.71%)	1.32 (1.02%)	2.34 (1.80%)	
Writing data to export buffer (sec)	0.298 (0.23%)	0.279 (0.21%)	0.312 (0.24%)	0.464 (0.36%)	0.26 (0.20%)
Network communication (sec)		2.16 (1.66%)		0.29 (0.22%)	1.33 (1.02%)
Total	1.45 (1.12%)	3.37 (2.59%)	27.622 (21.25%)	24.632 (18.95%)	2.538 (1.95%)

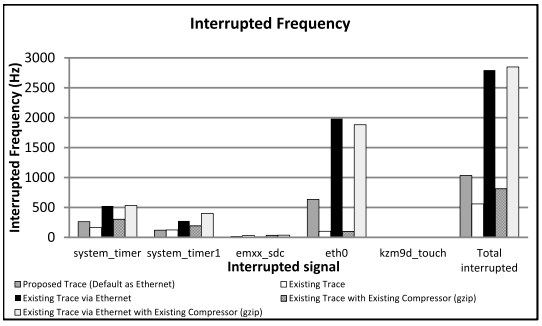

図 10 割込み頻度の比較
Fig. 10 Comparison of interrupted frequency.

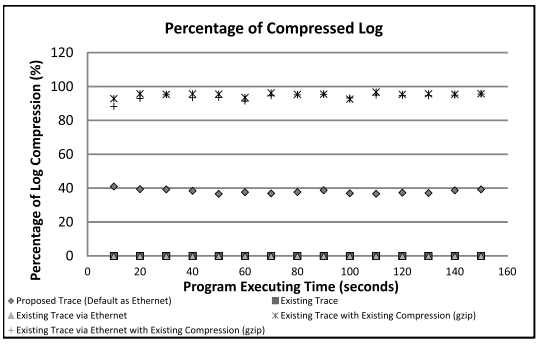

図 11 ログデータの圧縮率
Fig. 11 Percentage of compressed log.

スデータに変換した後，BSF ログのサイズと Ftrace 形式のサイズとを比較したものである．

実験結果は**図 11** に示す．提案した手法は多少の変動はあるものの，ほぼ 40%の圧縮率でトレースデータを圧縮できていることが分かる．

5) **システムのオーバヘッド（Overhead of System）**
提案手法を実現するために機能の追加，分離を行った際に，それぞれの機能の実行時間をシステムのオーバヘッドとして計測した．具体的には，ネットワークのセットアップ，圧縮，文字列化，データのエクスポートついて，130 秒間実行を 3 回行い，その平均値を取得した．

- **Network Communication** は，イーサネットによるデータ転送をセットアップするのにかかるオーバヘッドである．
- **Compressor** は，本提案方式のデバイス制御におけるデータを圧縮するのにかかるオーバヘッドである．
- **String Converter** は，従来方式での文字列化にかかるオーバヘッドである．
- **Writing Data to Export Buffer** は，エクスポートバッファにデータを書き込むのにかかるオーバヘッドである．

実験結果を**表 5** に示す．“Proposed Trace” は，“Existing Trace” よりもオーバヘッドが大きいが，“Existing Trace via Ethernet” と比較して，半分以下のオーバヘッドとなっている．特に，ネットワーク転送を行うためのセットアップのオーバヘッドが大幅に低減している．また，文字列化にかかるオーバヘッドが圧縮とほぼ同等の時間である．

4.3 考察

本章では，前章の実験結果について考察する．

4.3.1 エクスポートされたトレースデータのサイズとコンテキストスイッチの数

図 8 より，提案された手法は，データの増加率が既存の方法をイーサネットで送信することに比べて低く抑えられていることが分かる．これによって，システムが保持しなければならないトレースデータのサイズを押さえるとともに，外部に転送するときのネットワーク帯域を低く押さえることが可能となっている．

“Existing Trace via Ethernet” においてデータサイズが上方にシフトしているが，これは Trace pipe を用いることによるオーバヘッドと．ネットワークを介したデータ転送を必要とするためであると考えられる．

また，起動後 90 秒のあたりでデータサイズが小さくなる現象が起きている．これは図 11 からデータサイズの圧縮率が高くなっていることが分かる．これによってデータ

サイズが小さくなったことが原因であると考えられる．また，それ以降 “Existing Trace” と比較してデータサイズが小さくなっているが，本圧縮方式では前のデータに影響を受けることから，一度圧縮率が上がったことでそれ以降のデータサイズが影響を受けているものである．

図 9 から，コンテキストスイッチ数についても “Existing Trace via Ethernet” より小さな増加率となっていて，提案手法がコンテキストスイッチ数を低減させていることが分かる．このグラフでも，“Exisiting Trace via Ethernet” が上方にシフトしているが，この理由としてはユーザ空間へのデータのコピーとネットワーク転送ためのシステム呼び出しによってコンテキストスイッチが増加しているためだと考えられる．

また，30 秒〜80 秒，100 秒，130 秒以降でコンテキストスイッチの増加が見られるが，図 11 と比較すると圧縮率の変動に関連していることが分かる．これは，圧縮率が落ちている部分では，エクスポートする回数が増えることでネットワークの転送が必要となるので，その分コンテキストスイッチが増加しているものと考えられる．

4.3.2 “Trace Driver” の影響と圧縮機能の効果

表 5 を基に，データの特徴 (A)，(B)，(C) のオーバヘッドを減らす効果について考察する．システムのオーバヘッドについては，“Proposed Trace” での圧縮機能のオーバヘッドが，“Existing Trace” の文字列化とほぼ同等であると明らかになった．しかし，コンテキストにアプリケーション名の異なるデータへの切替えが頻発する場合，データの特徴 (A) の圧縮率が低下してしまう．一方，特徴 (B) と (C) は上記のデータに依存しないことから，一定の圧縮をすることは可能である．また，今回は未実装だが，(A) の部分で登場するプロセス名などの文字列は頻繁に変わらないことから，変換テーブルを用意し，ID を割り当てることで圧縮率を改善することも可能である．

また，130 秒の実行時間における本提案手法のシステムオーバヘッドが，2.39 秒と実行時間の 1.8%程度に収まっていることから，既存のトレース手法とほぼ同程度のオーバヘッドであり，実用上問題がないことが明らかになった．既存の手法をユーザ空間からネットワーク転送する場合と比較して，セットアップ時間が 1/3 程度に短縮されている．また，データサイズも 60%程度に縮小していることから，ネットワーク転送時間も低減させることが可能である．

4.3.3 割込み頻度とトレースデータとの関係

図 9 と図 10 から，割込みの頻度が多いほどコンテキストスイッチの数が多くなるので，トレースデータサイズが増えログ消失の可能性が高くなることが分かる．“Existing Trace” は割込みの頻度の合計が最小なので，コンテキストスイッチ数が最も少なく，1 秒当たりのトレースデータの生成のサイズが最小になる．その結果として，リングバッファが新しいデータで上書きされる可能性は最小になり，トレースデータ消失の可能性も最小になると考えられる．そして，“Existing Trace via Ethernet” のデータから，そのトレースデータをイーサネット経由でマシンの外部に転送する場合，同様に割込みの頻度の合計が最大になるのでトレースデータ消失の可能性も最大になると考えられる．

これに対して，本提案のトレース機構では，ネットワーク通信の割込み回数を “Existing Trace via Ethernet” の約 30%に削減し，トレースデータサイズを従来の 40%に圧縮して転送することが可能である．この理由としては，イーサネット経由で圧縮されたデータを外部に転送するときに，トレースデータのサイズを 40%に小さくした転送が可能であることと，カーネル内で転送処理を行うことによる割込み回数の低減が考えられる．ログデータを低減させることで，ネットワークに占めるログデータ転送に必要となる帯域を減らすことができる．これは，多くの機器がネットワークに接続される場合により効果的である．

タッチパネルの割込み（kzm9d_touch）については，実験結果によるとタッチパネルの割込みが発生していない．これは，抵抗膜式のタッチパネルなので固定の抵抗力にしか割込みが発生しないものである．本を載せた瞬間にタッチパネルの割込みが発生するが，実験中にはパネル上の本を移動させていないためである．

また，SD カード（emxx_sdc）の割込みについては，実験結果によると “Proposed Trace” と “Existing Trace via Ethernet” は割込みが発生している．これは，本実験環境の KZM-A9 評価ボードは SD カードからカーネルをおよび Android，ブラウザを起動することから，このブラウザプログラムによるページングによって，SD カードへの入出力が発生して，割込みが発生していると考えらえれる．

4.3.4 転送処理自体をカーネル空間内で処理したことの影響

通常 Ftrace はリングバッファに収集されたトレースデータを 100 m 秒ごとのタイマ割込みによってエクスポートするが，本実験環境では NO_HZ の指定による変動周期のため，タイマ割込みはシステムが busy か idle にかかわらず，必要な場合にだけ発生する．本実験の結果によるタイマ割込みは，図 10 の “system_timer” と “system_timer1” である．この結果から見ると，“system_timer” と “system_timer1” の差分は Ftrace の処理によるタイマ割込みであると考えられる．その場合，システムタイマの割込みの頻度を従来の 50%に削減したことが分かる．原因としては，Existing Trace via Ethernet においてユーザ空間に起動する図 1 の “Log acquiring program” 自体のディスパッチの回数の削減があげられる．そして，NO_HZ の指定のため，Ftrace の Trigger は 2 つの CPU コアのどちらかに偏って発生している現象が発生していることが考えられる．評価実験結果によると，Ftrace のデータエクスポートは CPU0 の方で主に処理されていることから，“system_timer” と

“system_timer1” とが異なった結果となっている．

一方，カーネルに転送処理を移行したことにより，カーネル自体の割込みがマスクされる可能性がある．実験ではタイマ割り込みが NO_HZ の指定による変動周期なので，システムが busy でも idle でもタイマ割込みは必要なタイミングでだけに発生する．タイマ割込みのマスクの頻度は，測定該当トレースの “system_timer”，“system_timer1” の合計と Existing Trace の “system_timer”，“system_timer1” の合計との差分で表すことができる．本提案手法の割込みのマスクは 98 回であり，Existing Trace via Ethernet の割込みのマスクは 515 回である．これは圧縮によるトレースデータサイズの低減，およびユーザ空間へのデータコピー回数の低減による効果である．この結果，本提案手法は，ユーザ空間でイーサネットを用いた場合と比較してマスクの回数を約 20%に削減したことが分かる．これにより，本方式がシステムの応答性の面でも改善されていることが分かる．

4.3.5 既存圧縮手法との比較議論

図 11 と表 5 から既存圧縮手法の gzip を用いる圧縮率が 95%程度である．そして，1,678,149 バイトの圧縮済のトレースデータの転送時間が 0.29 秒である．Gzip の圧縮オーバヘッドが 33 秒である．圧縮率が高くなるが，メモリの制約が厳しい組込み機器で圧縮したため，圧縮処理自体の影響でトレースデータのサイズが 30,937,475 バイトに増大する．1 秒当たりのトレースデータの生成サイズが 106,680,948 バイトなので，リングバッファからデータの収集効果が低減して，データの消失可能性が高くなる．

一方，本圧縮手法はカーネル空間から圧縮したことで，圧縮率が 40%程度であるが，圧縮オーバヘッドが 0.948 秒の小さいものである．そして，カーネル空間から直接に Log Acquiring Server に 10,380,256 バイトのトレースデータを転送する時間が 1.18 秒である．1 秒当たりのトレースデータの生成サイズが 8,796,827 バイトであり，既存圧縮手法より小さいことが分かる．なので，本提案手法はメモリ制約の厳しい組込みシステムにおいても，リングバッファからデータの収集効果が向上されたことでデータの消失可能性も削減することでトレースデータを利用することが可能であるので，40%の本提案手法の圧縮率妥当であると明らかになった．

4.3.6 Log Acquiring Server のオーバヘッド

“Log Acquiring Server” におけるオーバヘッドは，従来の Ftrace 形式と JSON 形式とがほぼ同等である．従来のトレース機構と同等のオーバヘッドで Log Monitoring Environment においてブラウザに適用するツールやアプリケーションを実装するのに有効である．

これらの考察から，本提案手法は，カーネル内に実装した低オーバヘッドの圧縮機能により，不要な割込みの頻発や割込みのマスクなどのシステムのスケジューリングに与える悪影響の要素を小さくしたことと Log Monitoring Environment におけるブラウザに適用するアプリケーションを実装するのにも有効であることで，ネットワーク接続された組込みシステムのトレース機構を利用することで，可視化アプリケーション [2] を作ったりするのに有効であることが明らかになった．

5. 関連研究

5.1 LTTng/LTTv

従来のトレース機構として，Linux Trace Toolkit Next Generation（LTTng）と Linux Trace Toolkit Viewer（LTTv）がある [5]．LTTng は標準のカーネルトレースのようにシステムコールを用いずに，ユーザ空間でロギングを制御する低オーバヘッドのトレース機構であり，Common Trace Format（CTF）に変換したトレースデータをユーザ空間にエクスポートする．これは機器自体にトレースデータ保存する形である．そして，SSL によってデータを送信し，ホスト側で起動する LTTv で解析して可視化するツールキットである．

CTF を利用することで複数のツールでトレースデータを利用することができる．また，トレース機構のオーバヘッドは低く，ユーザ空間にエクスポートし，カーネルを通して二次記憶上に保存する．しかし，LTTng では各ツールはユーザ空間で実行されることから，コンテキストスイッチへの影響を避けることはできない．また，ネットワークを介して複数のマシン間でトレースデータをやり取りする場合，ユーザ空間のツールを利用するために実行コストがかかるという問題がある．

5.2 TLV

Trace Log Visualizer（TLV）[6] は汎用性と拡張性を実現するのが目的とする可視化ツールである．汎用性を実現するためにトレースデータの標準形式を定め，これを変換ルールによってユーザが外部から指定した仕組みで提供する．汎用的な可視化表示機構を提供し，ユーザが可視化表示項目に合わせて拡張することができる．

これは，マルチコアをサポートした RTOS である TOPPERS を対象とした可視化ツールであり，組込み Linux カーネルに対しては現在サポートが行われていない．

5.3 SystemTap

SystemTap [4] はユーザ空間およびカーネル空間のイベントを取得する Linux のプローブツールである．これは，拡張性，使いやすさ，性能，透明性，簡潔性，柔軟性，安全性の中で，必要なものを選択して利用できる．カーネルデバッグユーザは特定のトラップスクリプト言語を書き込んでコンパイルすることによって特定のイベントを取得することができる．しかし，SystemTap 自体はユーザ空間で

動作することから，スケジューリングへの影響を生じる可能性があるので，コンテキストスイッチのディスパッチに影響を与えないトレース機構が必要である．

6. むすび

本論文では，組込み Linux システムを対象として低オーバヘッドなプロセスのトレース機構を提案し，実装評価を行った．システムはトレースデータ生成側の組込み機器とトレースデータ取得側の計算機とに分離し，カーネル内部でデータを転送することで，生成側への影響を減らしつつトレースデータを取得することが可能になった．トレース時にデータフォーマットの特徴を活かして，実用上問題がない低オーバヘッドの圧縮機構を組み込むことで，より多くのトレースデータをメモリ内に保持できるようになり，システムのリアルタイム性への影響を減らすことを可能にした．これを Linux システム上で実装評価を行い，本方式を，従来の Ftrace をメモリ上に記録した場合およびトレースデータをユーザ空間からネットワークで転送した場合と比較を行った．その結果，本提案方式は，平均で 40%のトレースデータの圧縮が可能であることが明らかになった．

結果として，ユーザ空間へのプロセス切替えによる割込みの頻度や割込みマスクの回数を減らしつつ，生成されるトレースデータのサイズを削減させたことで，システムのスケジューリングやネットワークの帯域などの影響を低減させつつトレースすることを可能にしたことから，システムにおけるトレース機構として有効であることが明らかになった．トレース機構として有効であることが明らかになった．

参考文献

[1] Maia, C. et al.: Evaluating Android OS for Embedded Real-Time Systems, Technical Report IPP HURRAY, HURRAY-TR-100604, pp.1–8 (2010).

[2] 中川裕貴, Praween, A., 西野洋介, 早川栄一：Development of a Visualization Enviroment for Android Operating System，研究報告組込みシステム（EMB），Vol.2013, No.30, pp.1–6（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009551230/en/⟩ (2013).

[3] Nakagawa, Y.: Development of Operating System Process Visualization Environment in Android, SIG Technical Reports (SE), 2011-SE-172, pp.1–6 (2011).

[4] EIgler, F.C. et al.: Architecture of systemtap: A Linux trace/probe tool (online), available from ⟨https://sourceware.org/systemtap/archpaper.pdf⟩ (accessed 2015-08-15).

[5] Desnoyers, M. and Dagenais, M.R.: The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux, *Linux Symposium*, Vol.1, pp.209–223 (2006).

[6] 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の開発（ドライバ，ツール，組込技術とネットワークに関するワークショップ ETNET2009），電子情報通信学会技術研究報告．CPSY，コンピュータシステム，Vol.108, No.463, pp.73–78（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110007324890/⟩ (2009).

[7] Sugaya, M. et al.: Online Kernel Log Analysis for Robotics Application, *Journal of Information Processing*, Vol.21, pp.53–66 (2013).

[8] LinuxDistributors, Ftrace (online), available from ⟨http://elinux.org/Ftrace⟩ (accessed 2016-11-18).

[9] RedHat, ftrace (online), available from ⟨https://access.redhat.com/site/documentation/ja-JP/Red_Hat_Enterprise_Linux/6/html/Developer_Guide/ftrace.html⟩ (accessed 2016-11-18).

[10] Microcomputer, K.: EM-EV2 Evaluation Board–KZM-A9-Dual–Hardware Operation Manual, Published with KZM-A9-Dual board (2010).

[11] song xian guang, ksocket (online), available from ⟨http://ksocket.sourceforge.net⟩ (accessed 2016-07-01).

プラウィーン アモーンタマウット

2012 年拓殖大学工学部情報工学科卒業．2014 年同大学院工学研究科電子情報工学専攻博士前期課程修了．現在，同博士後期課程に在学．システムプログラミングの研究を行う．

早川 栄一

1989 年東京農工大学工学部数理情報工学科卒．1991 年同大学院博士前期課程修了．1994 年同博士後期課程単位取得退学．博士（工学）．1994 年同大学電子情報工学科助手．1998 年拓殖大学工学部情報工学科助手．1999 年同専任講師．2003 年同助教授．2011 年同教授．オペレーティングシステム，組込みシステムを中心とするシステムソフトウェアの研究開発に従事．