Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

組込みシステム向け仮想化環境を用いた，汎用OSの監視機能

三浦 功也[1,a)]　本田 晋也[1]　高田 広章[1,b)]

概要：組込みシステムがネットワークに接続するにつれ，外部からの攻撃に対抗する手段が必要となってきている. 汎用 OS のみで防御を行う方法もあるが，リアルタイム性に欠けること，また，信頼性の確保が難しいという点から，組込みシステムへの適用は難しい．そのため本論文では，組込みシステム向け仮想化環境である SafeG を用いることにより，独立監視系 OS から汎用 OS を監視するための機能を作成し，その評価を行った．具体的には，汎用 OS の実行シーケンスを取得する機能と，汎用 OS で発生した割込み数を取得する機能を作成するとともに，その実行オーバーヘッドの計測を行った．その結果，オーバーヘッドの増加を抑えつつ，機能が実装できたことが確認できた．

キーワード：組込みシステム，リアルタイム OS，セキュリティ，仮想化

Monitoring general purpose operating system using Real-Time Embedded Virtualization

Takuya Miura[1,a)]　Shinya Honda[1]　Hiroaki Takada[1,b)]

1. はじめに

組込みシステムに用いられるプロセッサの性能が向上するにつれ，組込みシステムの分野においても，従来のリアルタイム OS ではなく Android に代表されるような汎用 OS を使用したいという要求が高まっている．汎用 OS を利用する主な理由の一つとして，ネットワークへの接続が容易に実現可能である点が挙げられる．組込みデバイスをネットワークに接続し，クラウド技術を用いて多用なサービスをユーザに提供できるためである．しかし，組込みデバイスに汎用 OS を用いるためには，二つの課題がある．一つ目は，リアルタイム性である．汎用 OS は一般的にソフトリアルタイムシステムである．そのため，従来のリアルタイム OS に比べると，リアルタイム性が守れない可能性が高くなる．組込みシステムにおいてリアルタイム性が守れない場合，最悪の場合には重大な事故につながる可能性がある．二つ目は，信頼性である．汎用 OS はそれ自体が大きいシステムであり，また，一般に広く普及しているため，潜在的なバグが混入している可能性があるとともに，悪意ある攻撃者の攻撃対象となる可能性が増加する．組込みデバイスが攻撃され，乗っ取られてしまった場合，使用者の個人情報が盗まれたり，命の危険にさらされる可能性がある．

前者の問題については，システムを汎用 OS ではなくリアルタイム OS で構築することによって解決が可能である．後者の問題については，汎用 OS 向けの仮想化環境を用いることによって，OS またはハイパーバイザからもう一方の OS を監視したする手法や，システムと独立したハードウエアを設置し，OS の動作をそのハードウエアから監視することで信頼性を高める手法が存在する．しかし，仮想化環境を用いる手法では，ハイパーバイザを経由してデバイスにアクセスするため，ゲスト OS に余分なオーバーヘッドが発生するという問題がある．また，仮想マシン自体が大きいプログラムであるため，変更を加えた際にその検証が難しくなってしまう問題もある．別のハードウエアを設置する場合は，その分だけコストがかかってしまい，コストの増加を招いてしまうという問題がある．そのため，組込みシステムにおいて，これらの技術をそのまま用いるのは望ましくない．

こうした問題に対応するため，我々の研究室では，組込みシステム向け仮想化環境である SafeG を開発してきた．SafeG を用いることで，リアルタイム OS と汎用 OS を同時に実行することができる．SafeG では，汎用 OS よりも

[1] 名古屋大学大学院 情報科学研究科
[a)] taku@ertl.jp
[b)] hiro@ertl.jp

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

リアルタイム OS を優先して実行するようになっているため，リアルタイム OS がもつリアルタイム性を保証することができる．また，仮想化環境であるという特徴を用いることで，汎用 OS 外から汎用 OS の監視を実現することができ，信頼性を向上させることが期待できる．

そのため本研究では，SafeG を用いたシステムにおいて，監視機能の実現に必要な機能を提供することを目的として，いくつかの機能を実現した．具体的には，汎用 OS で発生した割込み数の取得機能と，汎用 OS の実行シーケンスの取得機能を実現するために必要な機能を提供した．本研究によって，SafeG を用いたシステムにおいて，汎用 OS の信頼性を向上することができるようになり，組込みシステムにおける要件を保ったまま，システム全体の信頼性向上に貢献することができる．

本論文の構成は，以下のとおりである．まず，第 2 章において，現在の組込みシステムを取り巻く環境について述べるとともに，セキュリティの必要性について述べる．第 3 章では本研究で用いる SafeG を用いた監視機能について述べ，第 4 章では今回実現する監視機能の説明を行う．第 5 章で提供する機能を説明し，第 6 章では実現した機能の評価を行い，その結果について考察する．最後の第 8 章でまとめと今後の課題について述べる，

2. 組込みシステムの高信頼化

本章では，現在の組込みシステムが直面している脅威について説明する．その後，汎用 OS での対応の限界について説明した後に，SafeG を用いて組込みシステム上で監視機能を構築する際の要件について述べる．最後に，本研究の対象とした仮想化環境である SafeG について述べ，それらの要件が解決可能かについて議論する．

2.1 組込みシステムを脅かす脅威

近年の組込み業界は，スマート家電やスマートハウスといった，HEMS に関する製品が増加するとともに，自動車の多機能化といった動きもあり，業界全体としてネットワークへ接続を試みる傾向にある．HEMS の分野では，テレビなどの組込み機器自体が直接ネットワークに接続し，クラウド技術を利用することで様々な機能を提供する．自動車の分野では，車載ネットワークの Ethernet 接続や，クラウドを用いた自動運転を行う傾向にある．ネットワーク接続を提供するための機器には，接続を容易に実現するために汎用 OS が搭載されていることが多い．しかし，製品をネットワークに接続すると，その製品が外部からの攻撃にさらされることになるため，利用者の個人情報の流出や，製品の乗っ取りが起こるだけでなく，誤動作によって人命が危険にさらされたり，乗っ取った製品を用いて特定のシステムを攻撃する可能性があるため，乗っ取りをさせないためのセキュリティ対策が必須である．

2.2 汎用 OS のみでの対応の限界

汎用 OS で一般的に用いられるセキュリティ対策としては，ウイルス対策ソフトやパケットフィルタリング，IPS/IDS などが存在する．これらを導入することで，ウイルス特有の動作を検知してその動作を停止したり，想定していないパケットを破棄したり，外部からの攻撃や侵入を検知することが可能である．しかし，これらの対策をそのまま組込みシステムに適用することは望ましくない．理由として，汎用 OS は OS 自体が大きい点が挙げられる．汎用 OS では，定期的に OS 自体の脆弱性や OS 上で動作するソフトウエアの脆弱性が報告されており，リアルタイム OS に比べると信頼性が確保されているとは言い難い．また，汎用的に使われる USB や SD カードなどを備える可能性が高く，これらを経由した攻撃への対策が必要であり，それらを扱うソフトウエアの信頼性も必要である．これら全ての信頼性を担保することは難しく，大量のコストがかかってしまうため，信頼性が求められる組込みシステムへそのまま適用することは難しい．さらに，組込みデバイスには信頼性の他にリアルタイム性も求められる．一般的に汎用 OS はリアルタイム OS に比べてリアルタイム性の確保が難しい．そのため，汎用 OS のみで構成されたシステムの場合，そもそも組込みシステムに求められる要件の一つが満たせないことになる．そのため，汎用 OS のみでシステムを構築することは望ましいことではない．

既存のソフトウエアを用いる以外に，仮想マシンや外部のハードウエアを利用して，システムを提供するマシンの外から監視を行うことにより，信頼性を向上させる取り組みが存在する．この方法であれば，安全な外部の OS から監視を行うため，システムを提供する OS 自体にセキュリティホールがあっても検知が可能である．しかし，これらの仮想マシンは，デバイスを扱う際にハイパーバイザを経由するため，ゲスト OS のオーバーヘッドが大きくなってしまう．そのため，ゲスト OS としてリアルタイム OS を動作させる場合，リアルタイム性の確保が難しくなってしまう．

2.3 監視機能の要件

SafeG で監視機能を実現するための要件は，以下のとおりである．

(1) 検証容易性
　組込みシステムには信頼性が求められる．そのため，設計したとおりに動作することを保証するため，ソフトウエアの検証が十分になされなければならない．大規模なプログラムでは検証の際に不具合を取り除くことが難しくなるため，望ましくない．ソフトウエアの検証を容易に行えるようにするために，機能の実現は小規模なソースコードの変更量で実現する必要がある．

(2) 汎用 OS 非依存の実装

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

汎用 OS は大規模なプログラムであり，それ自体にバグが存在している可能性がある．監視機能実現のために汎用 OS への依存度が高い場合，攻撃者によって監視機能が無効化されてしまう可能性が高くなる．そのため，汎用 OS に出来る限り依存しない設計とする．

(3) 監視機能のマルチコア対応
組込み分野においても，近年使用されるプロセッサは，性能向上のためにマルチコア化が進んでいるものが多い．そのため，マルチコアでの動作を前提とした設計とし，必要に応じてコアの指定やロックの取得などの処理を行う．

(4) 短い実行時間
仮想マシンの実行時間が大きい場合，独立監視系 OS の処理時間に影響が出たり，汎用 OS の性能低下を招く．そのため，仮想マシン自体の実行時間は出来る限り短くする必要がある．

2.4 本研究の前提

本研究の前提は以下に示す通りである．

(1) システムはネットワークを介した攻撃を受ける
スマートデバイスなどに組み込まれたシステムが，物理的に攻撃を受ける可能性は否定出来ない．しかし，ネットワークに接続していることを考えると，ネットワークを介した攻撃を受けるほうが，物理的な攻撃よりも脅威となり得る．以上から，本研究では，ネットワークを介した攻撃を想定した防御機能の実現を目的とし，物理的な脅威は本研究の対象外とする．

(2) 汎用 OS はビルド時は安全である
プログラムを実装する段階では，実装者が意図的に悪意あるコードを埋め込むことは考えにくい．また，プログラム実装時に存在する脆弱性については，テスト時に排除されているべきであり，存在しないものとする．以上の理由から，汎用 OS ビルド時は安全であるとする．

(3) 汎用 OS はネットワークに接続するまでは攻撃を受けない
本研究は，汎用 OS を利用した組込みシステムがネットワークに接続することを想定している．そのため，汎用 OS が外部からネットワークに接続している時に攻撃を受けることを想定しているため，汎用 OS がネットワークに接続するまでは攻撃を受けないものとする．

2.5 SafeG

本研究では仮想マシンとして SafeG というソフトウエアを用いる．SafeG の概要を**図 1** に示す．SafeG は，組込みシステム向けの仮想化環境であり，従来のハイブリッド OS[2] を拡張し，仮想化環境を提供している．SafeG を用いると，1 つのターゲットボード上で，リアルタイム性と信頼性を確保したまま，汎用 OS とリアルタイム OS を安全に同時実行することができる．SafeG の実現には，ARM TrustZone と呼ばれる技術を用いており，ノンセキュア状態で汎用 OS またはリアルタイム OS を，セキュア状態でリアルタイム OS を動作させる．また，汎用 OS からは，リアルタイム OS が使用する資源にアクセスすることを禁止するとともに，リアルタイム OS を汎用 OS よりも常に優先的に実行する．こうすることで，リアルタイム OS を汎用 OS から保護しつつ，リアルタイム性を損ねることなくリアルタイム OS を実行している．SafeG はマルチコアにも対応しており [1]，その場合は，各コアごとに独立して OS のスケジューリングが行なわれる．

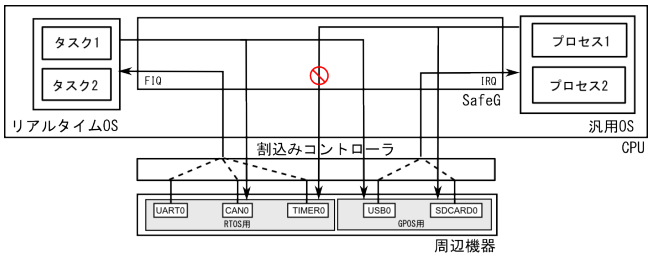

図 1 SafeG の概要

SafeG を用いることで，2.3 節の要件を解決することができると考えられる．検証容易性については，SafeG 全体のソースコードは，ターゲット依存部分を除けば 3919 行であり，一般的なプログラムに比べると，十分小さいと考えられる．また，独立監視系 OS としてリアルタイム OS が動作するため，これも汎用 OS に比べると十分小規模だといえる．そのため，本研究で実装する機能のソースコードが多くなければ，要件 1 については満たすことができる．SafeG は実行する OS を切り替える機能を持つのみであり，実行 OS を切り替える以外では，ゲスト OS の動作に割り込むことはない．そのため，監視機能が汎用 OS に依存することがなければ，要件 2 に関しても満たすことができる．また，SafeG はマルチコアにも対応しているため，マルチコア環境に対応したプログラムを作成することで要件 3 は満たすことができる．SafeG は実行 OS を切り替える以外の動作を行わないため，SafeG 自体の実行時間は短く，要件 4 も満たすことができる．

3. SafeG を用いた監視システムの構成

SafeG を利用した監視機構全体のシステム構成を**図 2** に示す．SafeG を用いたシステムでは，セキュア状態で制御系のシステムを実行し，汎用 OS でネットワークアクセスなどの，汎用的なシステムを実行する．それぞれのシステムは先述した TrustZone という技術を用いて独立しているため，汎用 OS が攻撃を受けた場合でも，その影響がリアルタイム OS には及ぶことはない．SafeG を用いてこうし

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

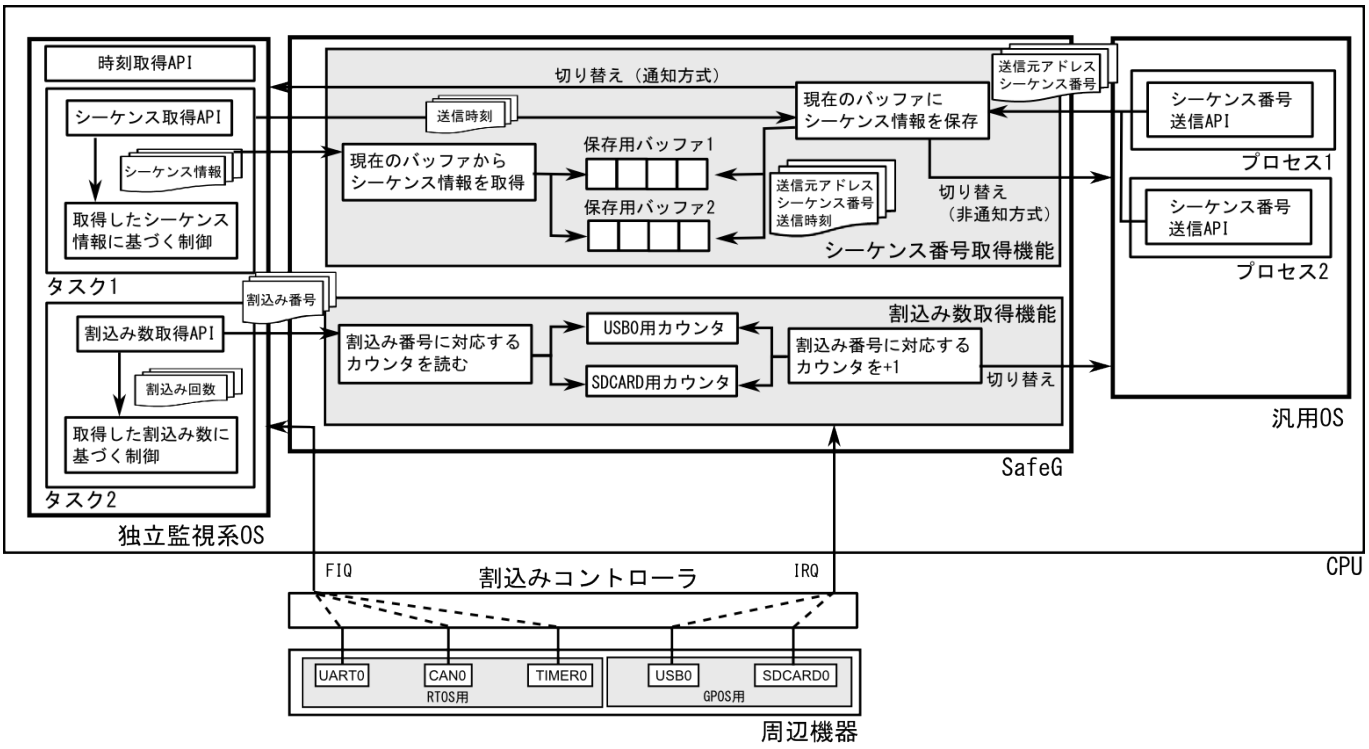

図 2 SafeG を用いた監視システムの概要

た監視を行うためには，汎用 OS とは独立した監視機能を持った OS（独立監視系 OS）をセキュア状態のリアルタイム OS として動作させればよい．SafeG の性質上，独立監視系 OS 自体は安全な領域にいることが保証されている．また，独立監視系 OS では監視機能と，その他必要最低限の機能を実行し，汎用 OS ではネットワーク接続をはじめとしたシステムを動作させることを考えると，独立監視系 OS がネットワークに接続することはなく，外部から一切の影響を受けない．そのため，独立監視系 OS は安全であるといえるため，独立監視系 OS から汎用 OS を監視することで，汎用 OS の信頼性の向上が期待できる．ネットワークに接続している汎用 OS が外部から攻撃を受け，何らかの障害が発生した場合でも，それを独立監視系から検知できれば，汎用 OS の動作を停止させ，独立監視系 OS のみで，必要最低限の機能を代行することができる．

4. 提供する監視機能

本研究では，提供する機能として実行シーケンスの取得機能と割込み数取得機能を採用した．それぞれの機能の概要を本章で述べる．

4.1 実行シーケンス監視機能

プログラムには，実行の流れが存在する．例えば，汎用 OS のカーネル初期化処理の場合，CPU の初期化，メモリ領域の初期化，ファイルシステムの初期化，割込みの初期化などを行い，最終的に特定の処理をループで実行する．監視部分固有のシーケンス番号を定義し，それを汎用 OS から独立監視系 OS に定期的に送信することで，こうした処理の流れを監視することが可能であり，該当する部分は実行されていることが確認できる．本研究の前提から，汎用 OS はビルド時は安全であり，かつ，ネットワークに接続するまでは攻撃を受けない．既存の監視機能にシーケンス監視機能を実装し，それをカーネルに組込んでビルドすることによって，既存の監視機能に変更がないことを確認できる．そのため，ビルド後にネットワークからの攻撃を受けた場合，それを検知することができるため，既存監視機構が意図したとおりに動作していることを確認することができる．

4.2 割込み監視機能

SafeG を用いたシステムでは，両 OS で発生した全ての割込みは一旦 SafeG でハンドリングを行った後に，目的の OS に渡される．そのため，割込み発生時に，発生した割込み要因を確認することが可能である．一定時間ごとに特定の割込み要因から発生した割込み数を確認することで，該当するデバイスの割込み機構が意図したとおりに動作していることを確認することができる．この機能を用いることで，汎用 OS の振る舞い検知型のウイルス対策ソフトウエアの実現が可能である．振る舞い検知型のウイルス対策ソフトウエアとは，ウイルスに感染した際の振る舞いの変化を監視し，攻撃を検出することである．攻撃の際には，何らかの入出力を伴う事が多く，デバイスにアクセスした

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

際の割込み数の監視が行える SafeG を用いることで，こうした機能を実現することができる．また，設計者が設計したとおりに割込み機能が動作しているかということの確認も行うことができる．

5. 機能の実装

本章では，先述した機能の実装に関して述べる．

5.1 提供機能の API

本研究で作成した機能の API 一覧を**表 1** に示す．

5.2 実行シーケンス取得機能

シーケンス番号の送信の際の振る舞いは，“通知方式” と “非通知方式” の二つの方法が選択可能である．通知方式とは，汎用 OS からシーケンス番号を送信する度に独立監視系 OS に切り替え，独立監視系 OS から確認する方式である．また，非通知方式とは，汎用 OS からシーケンス番号を送信した際，すぐには切り替えを行わず，一定回数分をまとめて独立監視系 OS から確認する方式である．

シーケンス番号は，safeg_seq（番号）を用いて汎用 OS から独立監視系 OS に送信する．それぞれの関数から safeg_syscall_seq_send を呼び出し，実際の番号の送信処理を行う．シーケンス番号の情報は SafeG 内部で管理され，汎用 OS からシーケンス番号が送られた時は，汎用 OS から送信されたシーケンス番号，送信時刻，送信時に汎用 OS が実行した関数のリンクレジスタの値を SafeG 内部に用意した保存用バッファに保存する．この保存用バッファの長さは静的に定義することができる．マルチコア環境では特定のバッファに同時にアクセスする可能性があるため，書き込み時に safeg_syscall_loc_seqbuf を呼び出すことで競合を防ぐ．ロック中に別のコアからシーケンス番号を送信された場合のことを考え，保存用バッファを多面とし，ロックする前に safeg_syscall_ch_seqbuf によって保存用バッファを切り替える．シーケンス番号の送信時刻を取得するために，SafeG 内部に関数ポインタを用意した．関数ポインタの参照先は，シーケンスの監視を始める前に safeg_syscall_reg_seqtimer を用いて設定可能であり，この関数からの返り値がシーケンス番号送信時の時刻として保存される．そのため，この関数についてはユーザが希望する精度において設定可能であるが，ユーザの責任において時刻を返す機能を実装する必要がある．

非通知方式を用いる場合は，シーケンス番号を保存する時は，保存用バッファが満たされない限り，独立監視系 OS には切り替えず，汎用 OS に戻ってその動作を続ける．これは，切替時にオーバーヘッドが発生することを考慮し，リアルタイム性を確保することを意図したためである．保存用バッファが一杯になった場合，専用のエラーコードを用いてその発生を独立監視系に通知するとともに，独立監視系 OS ではそのバッファの中身を確認する必要がある．そのため，独立監視系 OS に専用のエラーコードを返すことを目的として，独立監視系 OS から汎用 OS へ切り替えた時の返り値が保存してあるアドレスに，直接そのエラーコードを書き込んだ．通知方式を使用する場合は，シーケンス番号を送信する度に独立監視系 OS にその送信を通知する．通知方式は，非通知方式で使用するバッファの長さを 1 にすることで実現する．通知方式を使用する際は，非通知方式に比べてオーバーヘッドが大きくなることを考慮する必要がある．また，非通知方式の場合，用意したバッファが一杯になる前に，独立監視系 OS 側からシーケンス番号を確認することも可能である．その場合，意図しない時に独立監視系 OS に切り替えることを防ぐために，確認したバッファの内容を初期化する必要がある．しかし，バッファの内容を全て 0 で初期化する場合，実行時間が長くなってしまう．これを防ぐために，バッファのインデックスのみを初期化する方法を採用した．これにより，実行時間を短く保ったまま，バッファ全体の初期化と同様の機能を提供する．

5.3 割込み数取得機能

割込み要因ごとの割込み数を数える処理は，SafeG で行う．割込み番号を保存するために，全割り込み番号分の長さを持つ保存用バッファをコア数分用意する．IRQ が発生した際には，割込み発生コアに対応する配列の，割込み番号に対応する要素のカウンタを 1 上げる．割込み番号に対応した配列の要素は，割込み番号を識別子として判断する．つまり，割込み番号が 1 番の場合は配列の 1 番目の要素を，割込み番号が 2 番の場合は配列の 2 番目の要素をアクセス先として判断する．なお，割込みが FIQ の場合は，独立監視系で処理を行う割込みであるため，割込み数のカウントは行わない．

割込み発生回数の記憶に関して，特定の番号の割込み番号のカウントを個別に停止する機能は設けない．一つ目の理由として，全ての割込み番号を保存しても，メモリ使用量がそれほど多くないという点が挙げられる．デバイスの数やソフトウエア割込みの量にもよるが，今回の環境では，割込み番号は 200 から 300 程度であった．1 つの割込み番号のカウンタは 4byte で管理しているため，300 の割込み要因がある場合でも，1200byte 程度しかメモリを消費しない．マルチコア環境の場合は，これがコア数分必要になるが，4 コアの環境においても 5kbyte にも満たない．今回の研究で，本機能を実現したシステムに割り当てられているメモリ量は 1Gbyte であることから，この量は十分許容できる量だといえる．二つ目の理由として，割込み番号のカウントがオーバーフローしても動作に影響はないという点が挙げられる．仮に監視対象外の割込み数のカウントがオーバーフローしてしまっても，カウントが 0 に戻るだ

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

表 1 API 一覧

シーケンス監視機能	独立監視系用の API	safeg_syscall_get_seqbuffer	現在の保存用バッファの先頭番地とバッファ ID を取得する
		safeg_syscall_clear_seqbuffer	指定された保存用バッファの保存先を，そのバッファの先頭にする
		safeg_syscall_get_num_seq	指定された保存用バッファに保存されている情報の数を取得する
		safeg_syscall_reg_seqtimer	タイマ値取得用の関数を登録する
		safeg_syscall_loc_seqbuf	指定された保存用バッファをロックする
		safeg_syscall_unl_seqbuf	指定された保存用バッファをアンロックする
		safeg_syscall_ch_seqbuf	保存用バッファを切り替える
	汎用 OS 用の API	safeg_syscall_start_seq	シーケンスの監視を開始する
		safeg_syscall_stop_seq	シーケンスの監視を停止する
		safeg_syscall_seq_send	シーケンス番号を送信する
		safeg_seq（番号）	シーケンス番号送信用のライブラリ関数 （1 番を送信する場合は safeg_seq1 となる）
割込み数取得機能	独立監視系用の API	safeg_syscall_init_int	割込み数を初期化する
		safeg_syscall_get_int	指定された割込みが入った回数を取得する

けであり，監視していない割込み番号であれば独立監視系 OS の動作に影響はない．三つ目の理由として，オーバーヘッドの削減が挙げられる．個別の制御を行った場合，割込み番号を保存する処理の中で，発生した割込みのカウントをしないように設定されているかをその都度確認しなければならない．SafeG は，割込み発生時に必ず動作するソフトウエアであり，SafeG の動作が遅延すると，独立監視系 OS がリアルタイム性を守れなくなる可能性がある．そのため，実行時間の増加を防ぐために，こうした管理は行なわないこととした．ノンセキュア状態の割込みが発生してから実際に割込み番号を取得するまでの間に，セキュア状態の割込みが発生した場合はノンセキュア状態でもセキュア状態の割込みが発生したと認知してしまうが，これは SafeG の仕組み上防ぐことができないことであるため，機能としての対応は行わない．

割込み数初期化機能は，独立監視系 OS からのみ可能である．独立監視系 OS は，初期化したいコアの ID と初期化したい割込みの割込み番号を指定して，初期化機能を呼び出す．SafeG では，指定されたコアの配列の割込み番号で示された要素に保存してある値を 0 で初期化する．また，この機能は指定されたコアの全ての割込みを初期化する機能，全てのコアに存在する指定された割込み番号を初期化する機能，全てのコアに存在する全ての割込み要因の割込み数を初期化する機能も提供する．

マルチコアで動作する場合，特定のコアでバッファを読んでいる時に，別のコアでも書き込みを行う場合がある．更新結果に不整合が発生するため，こうした事象は防ぐ必要がある．本研究では，コアごとにロック変数を用意し，書き込み時にそのロックを取得する方法を採用した．書き込み前にロックが取得されているかを確認し，ロックが取得されている場合は，そのロックが開放されるまでスピンウェイトで待つことで，排他制御を実現する．

6. 評価

本研究での評価は**表 2** に示す環境で行った．評価内容としては，追加実装した後のコードサイズ増加量，各機能を追加したことによる実行オーバーヘッド，2.3 節で述べた要件の対応である．

表 2 評価環境

プロセッサ	Coretex-A9 MPCore
周波数	800MHz
メモリ	1Gbyte
独立監視系 OS	TOPPERS FMP カーネル
監視対象の OS	Linux (release 3.7.0)

表 3 コードサイズ

シーケンス番号取得機能	C ファイル	107 行
	h ファイル	4 行
割込み数取得機能	C ファイル	287 行
	h ファイル	15 行

6.1 コードサイズ

SafeG には，検証容易性が求められることから，機能の追加によるソースコードの増加量を求めた．増加量は，wc コマンドを用いて数えた．結果を以下の**表 3** に示す．

SafeG 本体のソースコードの中で，ターゲット依存部分を除いた全ての行数は 3919 行であった．SafeG 全てのソースコード量に比べると，今回追加実装した機能は 413 行であり，多くはないと言える．

6.2 実行オーバーヘッド

本研究で実装した二つの機能について，その実行時間を計測した．計測した時間は，割込み発生時に該当する割込み番号のカウンタを 1 上げるために必要な時間，シーケンス番号の送信を独立監視系 OS に通知する場合の実行時間，シーケンス番号の送信を通知しない場合の実行時間の三つである．実行時間の測定にはパフォーマンスカウンタを用いて行った．

割込み数取得時間

割込み数を保存するために，割込み処理の最初で割込み番号を保存するまでに必要な時間を計測した．その結果を**図 3** に示す．計測は割込み番号のカウンタを上げる処理の前後でパフォーマンスカウンタの値を取得し，その差分を

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

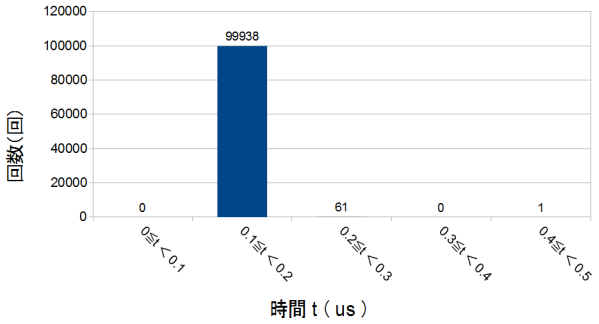

図 3　割込み数取得時間

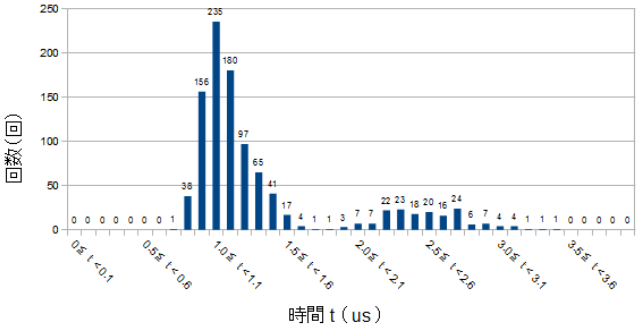

図 5　通知方式のシーケンス番号送信時間
（監視対象の OS として Linux を使用）

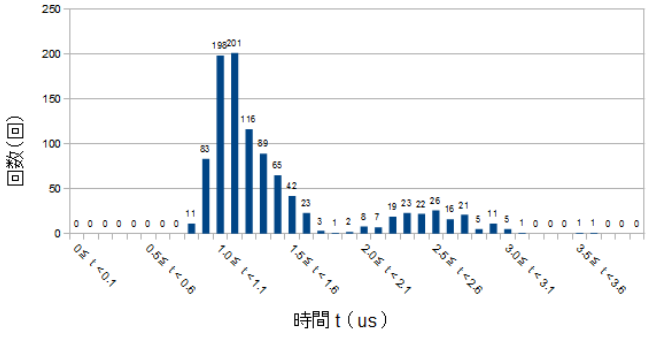

図 4　非通知方式のシーケンス番号送信時間
（監視対象の OS として Linux を使用）

取る形で 10 万回行った．

非通知方式のオーバーヘッド

監視対象の OS からシーケンス番号を送信し，独立監視系 OS に通知することなく戻ってくるまでの実行時間を測定した．監視対象の OS が送信用の関数を呼ぶ直前にタイマの値を読み，シーケンス番号の情報を保存する．その後，監視対象の OS に戻ってきた直後に再びタイマの値を読み，その差分を求めることでの計測を 1000 回行った．この計測結果は，非通知方式の場合に，バッファが満たされていない場合に該当する．監視対象の OS として，Linux を用いた場合の計測結果を**図 4** に示す．

通知方式のオーバーヘッド

監視対象の OS からシーケンス番号を送信し，独立監視系 OS に通知するまでの実行時間を測定した．監視対象の OS が送信用の関数を呼ぶ直前にタイマの値を読み，切り替えた独立監視系 OS の先頭でタイマを停止する．その後，戻ってきた監視対象の OS で再びタイマの値を読み，その差分を求めることでの計測を 1000 回行った．この計測結果は，通知方式の場合と，非通知方式の場合にバッファが満たされた時に該当する．監視対象の OS として，Linux を用いた場合の計測結果を**図 5** に示す．

考察

まず，割込み監視機能について考察する．本機能の計測結果では，ほぼすべての値が $0.1 \leqq t < 0.2$us の範囲に収まっていた．割込み監視機能を実現するための追加処理は，割込み番号を取得してカウンタを 1 上げるだけである．そのため，処理内容を考えるとこの実行結果は妥当だといえる．本計測では，初回の計測結果を無視する処理を行っていないため，$0.4 \leqq t < 0.5$us の範囲の結果は，初回測定結果のキャッシュミスだと考えられる．キャッシュのヒット，ミスに関わらず，今回の計測結果は 0.5us 以内に全て収まっている．IRQ 発生時のみにこの処理を行うことを考えると，Linux の実行時間がこの分だけ遅延する．Linux のスケジューラなどの基本機能が ms 単位で管理されていることを考えると，この時間の遅延はそれほど大きな影響を与えないと考えられる．

次に，シーケンス取得機能について考察する．SafeG が実行する OS を切り替え，再び元の OS に戻ってくるために必要な実行時間は 1.0us～4.0us である場合がほとんどであった．そのため，一度実行 OS を切り替えるための時間はその半分の 0.5～2.0us 程度である．今回の計測では，通知方式と非通知方式ともに 1us 前後の実行時間を示すことが多かった．監視対象の OS として FMP カーネルを用いて同様の計測を行った場合，このようなばらつきは起こらなかったが，1us あたりに計測時間が固まって存在していたことと，追加実装した機能の内容から考えると，実行時間として妥当だと考えられる．また，通知方式と非通知方式のどちらの場合においても，実行時間が 2.5us 程度まで延びてしまっている時が見受けられる．一般的に汎用 OS はリアルタイム OS よりもキャッシュへの依存度が高いため，この実行時間のばらつきは Linux のキャッシュミスによるものと考えられる．

6.3　要件との対応

2.3 節との対応について考察する．検証容易性については，6.1 の評価項目から，コード増加量は 1 割程度に抑えられており，十分満たせているといえる．汎用 OS への依存度に関しては，割込み数取得機能は汎用 OS に一切依存していない．シーケンス番号取得機能に関しても，汎用 OS には番号と LR の送信を依頼している．この点に関しては汎用 OS に依存してしまっているが，SafeG で別途タイム

Vol.2014-DBS-160 No.3
Vol.2014-OS-131 No.3
Vol.2014-EMB-35 No.3
2014/11/18

スタンプを取得することで，シーケンス番号の送信が送られなくなったり，攻撃者が番号を詐称して番号を送った場合にも検知が可能である．そのため，汎用 OS への依存度も低く設定されているといえる．また，シーケンス番号取得機能は保存用バッファを多重バッファとし，割込み数取得機能はコアごとに割込み数を取得することで，マルチコアへの対応も行った．よって，マルチコアへの対応の要件も満たしている．最後に，実行時間に関しては，6.2 節でも述べたように，許容できる大きさまで削減できている事がわかる．以上から，全ての要件を満たしていると考えられる．

7. 関連研究

組込みシステムを対象としたセキュリティアプローチは本研究以外にもいくつか存在する．文献 [5] では，汎用 OS のシステムコールのシーケンスを監視することで，攻撃検知を行っている．また，文献 [6] や文献 [3] は，実行バイナリから命令フローを何らかの形でグラフ化し，それを外部のハードウエアから監視することで，リアルタイムに攻撃検知を行っている．このように，組込み向けの監視では，監視対象のシステムを外部のハードウエアから確認することで，リアルタイムな攻撃検知を実現している．しかし，外部に別途ハードウエアを設けることになるため，その分だけ回路面積が増大したり，余分なコストがかかることになる．また，別の方法として，献 [4] のような方法がある．この方法では，仮想マシンを用いて，各仮想マシンにアクセス権を導入することによるアクセス制御を行っている．しかし，そもそも仮想マシンを用いた場合では，OS が直接ハードウエアを扱えないため，リアルタイム性に影響が出る可能性があり，組込みシステムにおける適用は懸念される．

8. まとめ

本研究では，組込みシステム向け仮想化環境である SafeG を用いて，独立監視系 OS から汎用 OS を監視するための機能を実装した．汎用 OS からシーケンス番号を送るために必要な機能と，汎用 OS で発生した割込み数を取得するために必要な機能の実装を行った．シーケンス監視機能では，汎用 OS から独立監視系 OS にシーケンス番号と LR の値を送信するとともに，SafeG で番号送信時刻を取得し，保存する．送信は通知方式と非通知方式を選択可能である．割込み数取得機能に関しては，汎用 OS 非依存で，発生した割込み要因ごとの回数を数えることで実現した．それぞれの機能のオーバーヘッドと，コード量に関して評価を行い，コード増加量，オーバーヘッド共に少ないことが確認できた．本研究では，あくまで監視機能の実現に必要な機能の実現を行っただけであり，監視機能自体の実現は行っていない．また，汎用 OS のシーケンス番号を監視するためには，シーケンス番号の送信はユーザモードではなく特権モードで行う必要がある．そのため，ユーザモードからシーケンス番号の送信を行えるように拡張することや，実際に監視機能を作成し，汎用 OS のアプリケーションを監視することなどが今後の課題である．

9. 参考文献

参考文献

[1] 太田貴也, Daniel Sangorrin, 一場利幸, 本田晋也, 高田広章“組込み向け高信頼デュアル OS モニタのマルチコアアーキテクチャへの適用”, 情報処理学会第 117 回 OS 研究会,(2011-4)

[2] 中島健一郎, 本田晋也, 手嶋茂晴, 高田広章“セキュリティ支援ハードウエアによるハイブリッド OS システムの高信頼化”, 情報処理学会研究報告 EMB, 組込みシステム, pp.1-7 (2008-11)

[3] Kekai Hu, Tilman Wolf, Thiago Teixeira and Russell Tessier “System-Level Security for Network Processors with Hardware Monitors”, DAC ’14 Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference (2014-06)

[4] Marcello Coppola, Miiltors D.Grammatikakis, George Kornaros and Alexander Spyridakis “Trusted Computing on Heterogeneous Embedded System-onChip with Virtualization and Memory Protection” CLOUD COMPUTING 2013(2013-5)

[5] Mehryar Rahmatian, Hessam Kooti, Ian G.Harris, and Elaheh Bozorgzadeh “Hardware-Assisted Detection of Malicious Software in Embedded Systems” IEEE EMBEDDED SYSTEMS LETTERS, VOL. 4, NO. 4(2012-12)

[6] Shufu Mao and Tilman Wolf “Hardware Support for Secure Processing in Embedded Systems” IEEE TRANSACTIONS ON COMPUTERS, VOL. 59, NO. 6(2010-6)