Vol.2013-SLDM-160 No.1
Vol.2013-EMB-28 No.1
2013/3/13

SafeGを用いた汎用OSの監視手法

三浦 功也[1,a)]　太田 貴也[2]　Daniel Sangorrin[2]　本田 晋也[2]　高田 広章[2]

概要：本研究では組込み向け高信頼デュアルOSモニタSafeGを用いた，汎用OSの監視手法を提案・実装した．SafeGは単一の組込みシステム上で，リアルタイムOS（RTOS）と汎用OSを同時実行するために提案・実装された小規模なソフトウェアモジュールである．SafeGを用いてRTOSから汎用OSを監視することにより，汎用OSのカーネルや，既存の監視機構が正しく動作していることを保証することができる．そこで本研究では，SafeGを用いて，RTOSから汎用OSの実行シーケンスの監視を行う機構に着目し，その実装と評価を行うことで，実際に汎用OSの監視機構が実現できることを示した．

キーワード：組込みシステム，リアルタイムOS，仮想化，セキュリティ，ルートキット

A method for monitoring GPOS using SafeG

Takuya Miura[1,a)]　Takaya Ohta[2]　Daniel Sangorrin[2]　Shinya Honda[2]　Hiroaki Takada[2]

Abstract: In this study, a method for monitoring a general-purpose operating system (GPOS) using SafeG, a highly reliable dual-OS monitor for embedded systems is proposed and implemented. SafeG is a small-scale software module designed for concurrently executing a GPOS and real-time operating system (RTOS) on top of the same embedded system platform. By using SafeG for monitoring the GPOS from the RTOS, it is possible to ensure that the GPOS kernel, as well as other existing monitoring mechanism, work correctly. Therefore, in this study, we focused on a method consist of execution sequence and developed and evaluated, we confirmed from evaluation results that this monitoring mechanism was successfully accomplished.

Keywords: Embedded system, Real-time OS, Virtualization, Security, Rootkit

1. はじめに

近年，組込みシステムにおいて，汎用OSをRTOSを同時に実行するために，組込み向けデュアルOSモニタSafeG [1][2] が開発された．SafeGは，ARM社のTrustZone [3] を利用することで，同一のハードウエア上で，RTOSと汎用OSを安全に同時実行しつつ，RTOSのリアルタイム性を確保している．

組込みシステムで汎用OSを使用する場合，汎用OSに対する攻撃手法への防御策が必要となってくる．また，Stuxnet [4] のように，制御システムをターゲットとしたウイルスも出現してきているため，制御システムにおけるセキュリティ対策が必要である．さらに，近年は組込みシステムのオープン化 [5] が進行し，機器や技術が標準化され，容易に接続・制御しやすくなっている．そのため，今後標準化された機器や技術が制御システムに広く使われるようになると考えられる．よって，組込みシステムへの攻撃が増加する可能性が高くなり，攻撃への対策がの重要性が増している．

汎用OSで外部からの攻撃を防ぐ方法としては，ファイアウォールやウイルス対策ソフトがよく知られている．しかし，セキュリティホールや特権昇格攻撃などにより，汎用OSのみで信頼性を確保することが難しい．汎用OSの信頼性向上のための方法として，汎用OSに依存しない信頼できる監視機構から，汎用OS上で動作している監視機構が正常に動作していることを確認する方法が考えられる．

[1] 名古屋大学工学部 電気電子情報工学科
Department of Information Engineering,Nagoya University
[2] 名古屋大学大学院 情報科学研究科
Graduate school of Information Science,Nagoya University
[a)] taku@ertl.jp

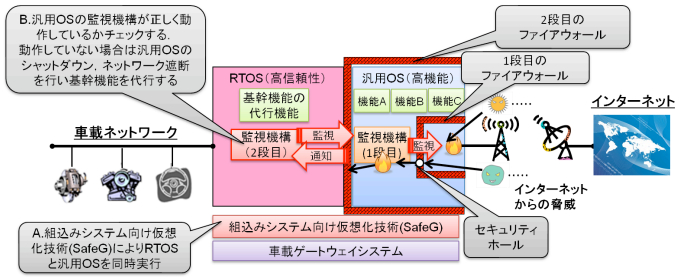

図 1 SafeG を用いた汎用 OS の監視
Fig. 1 Monitoring GPOS from RTOS.

SafeG を用いることで，信頼性が向上できる．**図 1** は，SafeG を用いた手法の全体像を示した図である．SafeG を用いることで，汎用 OS 上で動作している既存の監視機構を監視することができる．SafeG を利用したシステムでは，RTOS から汎用 OS の資源にはアクセスは可能であるが，汎用 OS から RTOS の資源にアクセスすることができない．そのため，RTOS が汎用 OS に攻撃されることはない．また，RTOS はネットワークに接続を行なわないため，ネットワークを介した攻撃も受けることがない．よって，信頼できる OS（RTOS）から汎用 OS を監視することが可能となる．

本研究では，SafeG を用いて汎用 OS を監視するフレームワークを構築した．まず，SafeG を用いた汎用 OS の監視機構が満たすべき要件を 5 つ示す．そして，SafeG を用いて RTOS から汎用 OS を監視するための機能の提案，実装，評価を行った．監視手法としてシーケンス監視を行う機構に着目し，その実現のために，RTOS から汎用 OS のシーケンス番号を確認する機能，汎用 OS のモジュールの正当性を確認する機能，汎用 OS に依存せずに検出を行う機能を実装した．また，監視プログラムを自動生成するためのコンフィギュレータの作成を行った．

実装した監視機構の評価として，SafeG のプログラムサイズ増加量，監視機能用システムコールのオーバーヘッド，RTOS の監視プログラム実行時間を測定した．その結果，監視機能用システムコールのオーバーヘッドは 31μs 以内，RTOS の監視プログラム実行時間は 23μs 以内，SafeG のプログラムサイズの増加量は，text セクションの 160 バイトであることが確認でき，5 つの要件を全て満たした監視機構を作成することができた．

2. 組込み向けデュアル OS モニタ SafeG

本章では，SafeG に利用されている TrustZone 技術と，SafeG のアーキテクチャについて述べる．

2.1 ARM TrustZone

TrustZone は，ARM 社が提供するセキュリティ拡張機能である．TrustZone 技術が実装されたプロセッサでは，セキュア状態とノンセキュア状態の 2 つの状態が存在する．セキュア状態では，すべてのリソースにアクセス可能であるが，ノンセキュア状態では，セキュア用と設定したリソースへのアクセスを禁止できる．汎用レジスタを含む一部のレジスタは各状態ごとに用意されており，それぞれの状態に対して特権モードと非特権モードが存在する．また，モニタモードと呼ばれるモードが追加され，セキュア状態とノンセキュア状態の 2 つの状態を切り替えるために使用される．モニタモードを呼び出すためには，SMC 命令を使用する．この命令が発行されると，SMC 例外が発生し，モニタモードのベクタテーブルにジャンプ後，その中に記述された処理を行う．

ARM の割込みには，FIQ（高速割込み）と IRQ（通常割込み）の 2 種類が存在する．FIQ は専用のレジスタを持っており，割込み処理の際に行うレジスタ退避などにかかる時間が IRQ よりも短い．また，FIQ と IRQ のそれぞれを個別に禁止することができる．

2.2 SafeG

SafeG は，汎用 OS と RTOS を安全に同時実行するソフトウエアモジュールである．SafeG はモニタモードで動作し，2.1 節で説明した TrustZone 技術を用いて，ノンセキュア状態で汎用 OS，セキュア状態で RTOS を動作させる．これにより，RTOS で使用するリソースへの汎用 OS からのアクセスを禁止できるため，2 つの OS の独立性を保ちつつ，信頼性の高いシステムを構築できる．

SafeG では，RTOS のリアルタイム性を保証するために，RTOS のすべての処理が，汎用 OS より優先実行される OS スケジューリング方式を採用している．従って，汎用 OS の処理により，RTOS の処理が阻害されることはない．RTOS または汎用 OS から SMC 命令を発行することで，SafeG を呼び出すことができる．これを SafeG システムコールと呼ぶ．例えば，OS 切り替えを行う SafeG システムコールでは，SMC 例外のベクタテーブルで，現在動作している汎用 OS のコンテキストを，汎用 OS のコンテキスト保存領域に保存し，RTOS のコンテキストを，保存した RTOS のコンテキスト保存領域から復帰する処理を行う．その後 SMC 例外から復帰すると，復帰した RTOS の環境で処理を再開することができる

3. SafeG を用いた汎用 OS の監視

本章では，SafeG を用いた汎用 OS の監視手法に必要な要件と，提案した監視手法について述べる．

3.1 汎用 OS 監視機構の要件

SafeG を用いた汎用 OS の監視機構が満たすべき要件は以下のとおりである．

要件 1 RTOS のリソース保護

RTOS のメモリ領域が破壊されてしまうと，機器の誤作

動や，RTOS のクラッシュを招くことになる．この場合，監視機構が動作する RTOS の信頼性を保つことができなくなるため，汎用 OS や RTOS に行った変更点により，RTOS の使用するメモリ領域が破壊されてはならない．

要件 2 RTOS の時間保護

SafeG は組込み分野での利用を対象としているため，RTOS のハードリアルタイム性は必ず守られなければならない．そのため，汎用 OS の漢詩紀行を実現するための変更により，RTOS の実行が遅延することはあってはならない．

要件 3 汎用 OS の最小限の変更

汎用 OS は複雑であり，また，頻繁にバージョンアップが行われる．そのため，監視を行うために必要な汎用 OS の変更点が多くなってしまうと，検証が難しく，また，バージョンアップへの対応が難しくなる．そのため，汎用 OS への変更点は少ないほうが望ましい．

要件 4 検証容易性

SafeG と RTOS にバグが存在すると，システムの信頼性を保つことが困難となる．そのため，テストによりそのバグの検出を容易とするために，SafeG と RTOS のコードサイズの増加はできる限り抑えることが望ましい．

また，汎用 OS は大規模であり，監視対象となるモジュールも多くなると考えられる．そのため，監視対象の変化や増加にも容易に対応できるように，ユーザによる機能追加や修正は，RTOS，汎用 OS ともに容易であることが望ましい．

要件 5 汎用 OS への非依存性

漢詩紀行の信頼性を保つためには，汎用 OS が攻撃を受けたり，予期せぬ動作をした場合にも，監視機構が動作を停止せず，その障害を検知できることが必要である．そのため，汎用 OS の監視機構は，汎用 OS に非依存であり，独立に動作することが望ましい．

3.2 SafeG を用いた汎用 OS の監視手法

本節では，汎用 OS を監視するための手法について述べ，その後，選択した手法について記述する．

3.2.1 シーケンス監視

一つ目の方法は，実行シーケンス [6] の監視である．実行シーケンスと呼ばれるプログラムの流れを監視することで，監視対象が正しい順序で実行が行われているかを監視することができる．汎用 OS から RTOS に対して，現在実行している部分に対応するシーケンス番号を送信し，その番号が正しいかを RTOS で確認を行うことで，正しく実行しているかを判断する．

3.2.2 チェックサム

二つ目の方法は，監視対象のモジュールのチェックサムを計算する方法である．あらかじめ対象モジュールのチェックサムを計算し，その値が変化していないかを周期的に観察する．チェックサムは比較的容易に実装ができ，その処理も単純なものであるが，前後の数字を書き換えることで比較的容易に改ざんを行うことができる．

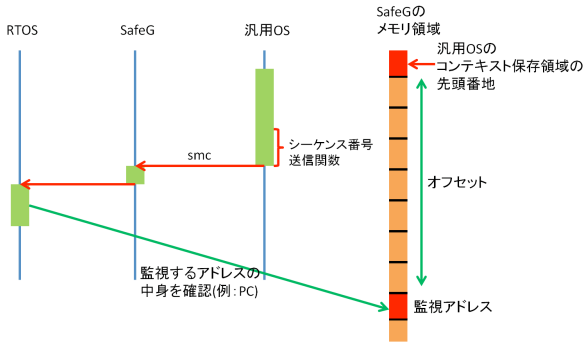

図 2 監視の流れ
Fig. 2 Monitoring flow.

3.2.3 割込み頻度の監視

SafeG を用いたシステムでは，割込みは全て SafeG でハンドリングを行い，決められた OS に処理を移す．そのため，汎用 OS でどの程度割込みが発生したかということは，SafeG で全て知ることができる．汎用 OS が攻撃を受けると，普段と異なる割込みが発生することが考えられるため，このような監視方法も有効であるといえる．

3.3 選択手法とその理由

SafeG を用いた汎用 OS の監視手法として，本研究では RTOS による汎用 OS のシーケンス監視手法を選択した．理由としては，汎用 OS での改ざんを行いにくいことと，監視機構構築の容易さ挙げられる．チェックサムの場合は，汎用 OS が攻撃を受けた時に，そのモジュールの改ざんが容易なため，RTOS からの検出が困難である．また，割込み頻度の監視の場合は，割込み頻度の変化をどの程度許容するかという点で，監視機構の構築が難しい．そのため，本研究ではシーケンス監視手法を採用した．

4. シーケンス監視機構の設計

シーケンス監視の流れは**図 2** のとおりである．まず，RTOS の初期化時に SafeG を呼び出し，監視を行うためのベースアドレスを取得する．汎用 OS は RTOS に現在の実行箇所を表すシーケンス番号を送信する．RTOS はシーケンス番号を受け取り，初期化時に得られたベースアドレスを基にして，正しい箇所から呼ばれているかを判断することで，汎用 OS の監視を行う．シーケンス監視を用いることで，3 節の要件を満たすことができると考えられる．理由としては，汎用 OS から RTOS にシーケンス番号を送信し，その値を RTOS から確認する作業を行うため，監視に必要な作業が少ない．よって，コードサイズが小さくなると考えられる．また，シーケンス番号を入手するために，RTOS から SafeG を呼び出す必要がないため，オーバー

ヘッドも小さくなると考えられる．よって，要件 3，要件 4，要件 5 を満たすことができると考えられる．また，要件 6 に関しては，タイムアウト機構を構築することで可能となると考えられる．実際に満たすことができたかどうかについては，評価の段階で確認を行う．

シーケンス監視の実現のために，汎用 OS からシーケンス番号を送信する必要がある．本研究では，シーケンス番号を RTOS に通知する方法として，RTOS で用意した変数領域に番号を保存する方式を提案し，設計，実装を行った．以下，この提案方式を実現するための汎用 OS，RTOS，SafeG への機能追加について述べる．

4.1 汎用 OS によるシーケンス番号の送信

提案手法では，汎用 OS から SafeG システムコールを発行すると，SafeG を介して RTOS まで切り替えを行う．その際に，SafeG システムコールの判別情報と，シーケンス番号を必要とする．そのため，汎用 OS から RTOS にシーケンス番号を送る SafeG システムコールである，safeg_seq_snd を作成した．この関数を発行すると，SafeG での OS 切り替えの際に，RTOS に用意したシーケンス監視用の変数に，汎用 OS から送信されたシーケンス番号を保存する処理を行い，RTOS に切り替える．その後，RTOS からその値を参照することで目的のシーケンス番号を得る．また，監視対象となるモジュールが複数存在する場合に対応するため，シーケンス番号と同時に，RTOS で処理すべきタスクの ID を送信する．RTOS で監視を行うタスクは，それぞれ汎用 OS の監視すべきモジュールに対応しており，汎用 OS から送信されたタスク ID を元に処理すべきタスクを決定する．

4.2 RTOS によるシーケンス番号の監視

RTOS から汎用 OS の監視を行うために，RTOS では，処理すべきタスク ID の確認，送信されたシーケンス番号の確認，RTOS に制御が移った際の汎用 OS のプログラムカウンタ（PC）の値の確認，シーケンス番号の送信回数の確認を行った．汎用 OS から送信されたタスク ID から，処理すべきタスクを起動する必要があった．そのために，制御用のタスクと，実際に監視を行うタスクを作成した．制御用タスクが，タスク ID に対応した監視用タスクを起動することで監視を行う．RTOS から汎用 OS の PC の値を入手するには，SafeG を呼び出すことにより行う．OS 切り替え時に，SafeG に保存した汎用 OS のコンテキスト保存領域の先頭アドレスを入手し，PC の保存領域までのオフセットを加える．このアドレスにアクセスすることで，RTOS に切り替わった際の汎用 OS の PC を参照するその値が，汎用 OS の実行ファイルに含まれている，safeg_seq_snd の先頭アドレスに SMC 命令までのオフセットを加えた値となっているかを比較することで監視を行う．

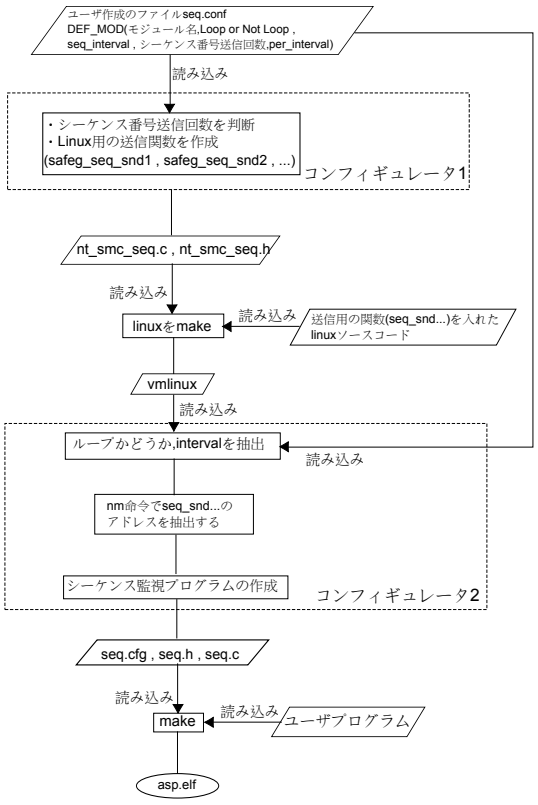

図 3 監視機構のコンフィギュレーション
Fig. 3 Configuration flow for monitor system.

4.3 SafeG システムコール処理の追加

提案手法を実現するためには，RTOS に汎用 OS のコンテキスト保存領域の先頭アドレスを返す機能と，汎用 OS から RTOS に切り替える際に，シーケンス番号とタスク ID を RTOS の変数に保存する機能が必要である．そのため，SafeG 内部にこれらの SafeG システムコールの機能を実現するための処理を追加した．それぞれの SafeG システムコールに，他のシステムコールとは異なる，ユニークな判別情報を与える．SafeG システムコールが発行された際の判別情報が，シーケンス監視用のシステムコールの判別情報であれば，それぞれに対応した処理を行う．

4.4 監視機構の自動コンフィギュレーション

コンフィギュレーションの流れを**図 3** に示す．ユーザは監視に無関係な RTOS アプリケーションを作成し，汎用 OS に safeg_seq_snd を挿入する．その後，監視対象のモジュールの情報を seq.conf に記述し，コンフィギュレータを動作させることで，RTOS 用のプログラムを自動的に生成する．監視モジュールはユーザプログラム[*1]とは独立しており, ユーザは監視モジュールを使用するときには, ユーザプログラムに監視モジュールをインクルードする記述を追加するのみでよい．ユーザが作成した RTOS アプリケーションのリアルタイム性を確保するため，監視機構は低優

[*1] 監視機構とは関係がなく，ユーザが自由に作成することができるプログラム

表 1　SafeG のプログラムサイズ
Table 1　Code size of SafeG.

	text	data	bss
監視機能なし	2,476	0	464
監視機能あり	2,636	0	464

先度タスクとして生成する．

5. 評価

評価環境として，プロセッサとして ARM Coretex-A9 シングルコアを搭載した R-Mobile A1 を搭載した評価ボード [7] を用いた．周波数は 800MHz であり，命令/データキャッシュは 32KByte，L2 キャッシュは 256KByte,RAM は DDR SDRAM 512MByte を備えている．実行時間の計測を行う評価では，ボードに搭載されたハードウエアタイマを用いた．評価用 OS として，RTOS には TOPPERS/ASP カーネル [8], 汎用 OS には Debian GNU/Linux（kernel:linux-2.6.35）を用いた．評価として，SafeG のプログラムサイズ増加量，監視機能用システムコールのオーバーヘッド，RTOS の監視プログラムの実行時間，の測定を行った．監視機能用システムコールのオーバーヘッドの測定と，RTOS の監視プログラムの実行時間の測定の際には，監視対象として，Liux 内部の scheduler_tick を選択した．この関数はスケジューリングを行うための関数であり，一定周期で起動させられる関数である．これは汎用 OS 上の監視機構ではないが，カーネル内部で動作をする関数であるため，監視を行うことで，カーネルの一部が正しく動作しているかを保証することができる．scheduler_tick の一回の実行ごとに，合計 5 回数字を送っており，5 回目の数字が送信された際に，送信回数が正しいかを確認する処理が行われている．

5.1 SafeG のプログラムサイズ増加量

要件 4 を満たすために，プログラムサイズを小さくする必要がある．そのため，監視機能用のコードを追加する前と追加したあとでの SafeG の実行ファイルである safeg.elf のプログラムサイズ増加量を測定した．測定結果を**表 1** に示す．size コマンドの結果から，手法実現のための機能追加により，text セクションのサイズのみが 160Byte 増加した．SafeG への機能拡張において，SafeG 内部に変数や定数を必要としなかったため，data セクションと bss セクションが増加しなかったと考えられる．評価ボードで使用可能な RAM 領域が 512MByte であることを考慮すると，プログラムサイズ増加量は十分小さく出来たといえる．

5.2 監視機能用システムコールのオーバーヘッド

要件 6 より，監視機構のオーバーヘッドはできる限り小さいことが望ましい．監視機構の導入により，汎用 OS の性能にどの程度影響を与えるかを測定するため，Linux から safeg_seq_snd を呼び出し，再び Linux に処理が戻るまでの時間を測定した．監視に無関係のアプリケーションとして，hackbench を選択し，hackbench の実行開始から終了までの間で計測を行った．なお，計測時に動作させた hackbench のスレッド数は 4000 とした．

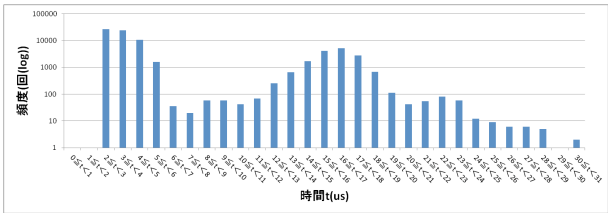

図 4　監視機能用システムコールのオーバーヘッド
Fig. 4　Overhead for monitor system.

測定結果を**図 4** に示す．図の横軸は，測定時間（μs）である．縦軸に頻度を表示しており，ログスケールで表示を行っている．結果の表示のために，計測結果を 1μs 単位で正規化をしている．

測定では，合計で 78985 回の測定結果が得られ，そのうち 5μs 以内に再び RTOS から汎用 OS に制御が戻ってきた回数が 61378 回だった．途中，14μs から 18μs かかった時間が 13770 回となっている．合計で約 80000 回計測しており，そのうち約 14000 回時間が延びていることを考えると，この部分の遅延はその回数判定が行われたために起こったものと考えられる．また，30μs など，それ以上かかっている部分はキャッシュミスが原因と考えられる．

5.3 RTOS の監視プログラム実行時間

要件 6 を満たすためには，RTOS のオーバーヘッドはリアルタイム性に影響を与えない程度である必要がある．RTOS での監視処理にオーバーヘッドがかかる場合には，他のユーザプログラムのリアルタイム性に影響を与えてしまうことが考えられる．そのため，RTOS の監視プログラムの実行時間を測定した．監視機能用 SafeG システムコールによって，RTOS に制御が移ったときに測定を開始し，シーケンス監視の全ての処理が終了した直後に測定を終了した．また，この測定も，hackbench の実行開始から終了までの時間の測定を行った．監視モジュールの一回の実行ごとに，シーケンス番号の送信を 5 回行うため，5 回に一度この処理が行われる．

測定結果を**図 5** に示す．今回の測定では，ほぼ全ての場合において 2μs 以内に実行が終わっていることが確認できる．今回の測定では評価 2 と同様に，5 回に 1 回シーケンス番号の送信回数の確認が入る．評価 2 と同様に，10μs から 13μs に頻度が集中していることから，この部分の遅延がシーケンス番号送信回数の確認処理によるものと考えられる．最悪実行時間は 23μs であり，リアルタイム性の確

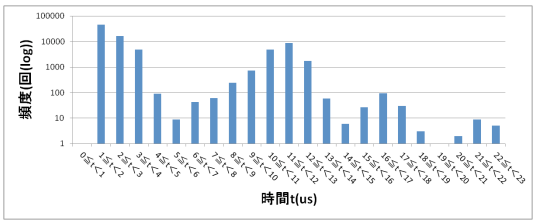

図 5 RTOS の監視プログラム実行時間
Fig. 5 Monitor execution time in RTOS.

保が著しく困難となるようなオーバーヘッドではないといえる．

5.4 要件との対応

本研究で実装した機構は，SafeG を利用して RTOS と汎用 OS を制御しているため，汎用 OS から RTOS のメモリ領域を破壊することはない．また，専用の API を作成しているため，実装前の機能の変更はしていない．そのため，シーケンス監視機構によって，実装前の RTOS の動作に影響を与えることはなく，要件 1 は満たしている．また，先の 5.3 節から，RTOS の最悪実行時間は 23μs である．さらに，監視タスクは最低優先度で動作するため，数μs 精度で制御が必要なタスクが存在する場合は，制御用のタスクよりも優先度を上げることで対応できる．そのため，要件 2 も満たしている．シーケンス監視を行う際に，汎用 OS に行う作業は SafeG システムコールを挿入するのみである．よって，実装の際の Linux への変更点は必要最小限になっているといえ，要件 3 も満たしている．また，自動コンフィギュレーションを行ったことにより，監視機構実現のためのコードの変更点は最小限に抑えることが確認できた．汎用 OS でのモジュールの追加や削除に関しても，監視モジュールに送信関数を挿入し，seq．conf を編集することにより，改変を行うことができる．コードサイズについても，5.1 節で述べたように，160 バイトと十分小さく出来た．そのため，複数の汎用 OS モジュールの監視への対応は十分可能であるといえる．よって，要件 4 も十分満たすことができた．タイムアウト機能を実装により，汎用 OS に異常が発生した際にも，監視機能が停止することがない．そのため，汎用 OS への依存度は少ない．また，タイムアウト機能は，アラームハンドラを用い，その処理のなかで，異常発生の処理を行うタスクを起動している．異常発生の処理を行うタスクは他のタスクよりも高い優先度で実行される．よって，タイムアウト機能は正しく実装できたと言え，要件 5 に関しても満たしている．

以上より，すべての要件を満たしている．

6. おわりに

組込み向けデュアル OS モニタ SafeG を用いて，RTOS による汎用 OS の監視機構を構築することができた．監視手法としてはシーケンス監視を用い，汎用 OS から送られたシーケンス番号が一致しているか，RTOS に制御が移った際の PC の値が正しいか，シーケンス番号の送信回数は正しいかという確認を行うことで，汎用 OS における実行シーケンスの異常検出を行った．

今後の課題として，現在は，静的に生成されたシーケンス番号送信関数にしか対応することができておらず，動的モジュールによって障害が発生してしまった場合は検知することができない．近年の多くの rootkit[9],[10] はこの形となってきており，監視機能強化のためにはこの拡張が不可欠である．今後は，動的モジュールへの対応や，汎用 OS のオーバーヘッドの削減，特権昇格攻撃などへの対応を行っていく予定である．

参考文献

[1] 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)
[2] 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)
[3] ARM 社 “ARM Security Technology PRD29-GENC-009492C”
[4] IPA “2010 年度 制御システムの情報セキュリティ動向に関する調査報告書”, 入手先 (http://www.ipa.go.jp/security/fy22/reports/ics_sec/documents/ics_sec2011.pdf)
[5] IPA“2011 年度自動車の情報セキュリティの動向に関する調査”, 入手先 (http://www.ipa.go.jp/security/fy23/reports/emb_car/documents/embsec_car2012.pdf)
[6] “JIS 規格規格番号 JISZ8116 自動制御用語 - 一般”, 入手先 (http://www.jisc.go.jp/app/pager?id=967251)
[7] “Armadillo-800EVA 製品マニュアル”, 入手先 (http://armadillo.atmark-techno.com/files/downloads/armadillo-800-eva/document/armadillo-800-eva_product_manual_ja-1.3.1.pdf)
[8] TOPPERS プロジェクト “TOPPERS 新世代カーネル統合仕様書 Release 1.4.0” 入手先 (http://www.toppers.jp/docs/tech/ngki_spec-140.pdf)
[9] Jeffrey Bickford, Ryan O’ Hare, Arati Baliga, Vinod Ganapathy, Liviu Iftode “Rootkits on Smart Phones:Attacks, Implications and Opportunities”, HotMobile ’10 Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, pp.49-54 (2010-02)
[10] Alkesh Shah,“Analysis of Rootkits:Attack Approaches and Detection Mechanisms”, 入手先 (http://nicolascormier.com/documentation/security/RootkitsReport.pdf)