2008－EMB－10 (1)
2008／11／27

セキュリティ支援ハードウェアによるハイブリッドOSシステムの高信頼化

中嶋 健一郎† 本田 晋也† 手嶋 茂晴† 高田 広章†
†名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター
Email: nakajima@nces.is.nagoya-u.ac.jp, honda@ertl.jp, teshima@is.nagoya-u.ac.jp, hiro@ertl.jp

概要

近年，車載情報システムは高機能化が著しく，情報系と制御系という異なる要件のアプリケーションを単一のシステムで実行したいという要求がある．この要求を満たすには制御系OS（リアルタイムOS）と情報系OS（汎用OS）をシングルプロセッサで並列に実行する必要がある．複数のOSを並列に実行する方法としてはハイブリッドOS方式とVM方式が提案されている．しかしながら，ハイブリッドOS方式は，制御系が情報系から保護されないという問題がある．また，VM方式を効率的に実現するためには，ハードウェア機構（VM支援機能）が必須であるが，一般的に入手可能な組込み向けのプロセッサでこの機能をもつものは存在しない．そこで我々は，組込み向けプロセッサが持つセキュリティ支援ハードウェア（TrustZone）を利用して，ハイブリッドOS方式を高信頼化した．具体的には，車載情報システムの要件を定め，要件を満たすようにTrustZoneを制御してハイブリッドOS方式を実現するソフトウェアモジュール（SafeG）を設計・実装した．そして，SafeGが定めた要件を満たしていることを実機を用いて評価した．

Enhancing reliability in Hybrid OS system with security hardware

Kenichiro Nakajima† Shinya Honda† Shigeharu Teshima† Hiroaki Takada †
† Center for Embedded Computing System, Nagoya Univ.

Abstract Next generation in-vehicle information systems are supposed to run on single platform on that both of two different domains; information systems and control systems, are integrated. Control systems require higher qualities in terms of dependability and real time real-time capability than those of information systems. This paper presents a software module SafeG, which allows dual OSes to run in parallel on a single processor and to keep security and real-time ability of control systems against insecure information systems. Design and implementation of SafeG based on ARM architecture is also discussed in the paper. TrustZone, which is secure extension in ARM processor is a key mechanism to implement SafeG functionalities.

There are two ways in order multi OSes to run in parallel. One is Hybrid OS system and another is VM system. In Hybrid OS system, there is no way to fully protect control system from information system. In VM system, there is also no common embedded processor with VM support function to effective execution. Our design allows to entirely protect the control systems from the information systems with less effort using common hardware in embedded systems. We demonstrate that our design has less overhead and enough realtime ability on a real hardware.

1 はじめに

近年，車載情報システムは高機能化が著しく，地図・道路情報をユーザーに提示するカーナビゲーションの機能に加えて，テレマティクスやインターネット接続といった機能を持つ機種もある．これらの情報系のアプリケーションに加えて，駐車・運転支援システムといった，地図・道路情報を利用してステアリングやエンジン等の制御系システムと連携する制御系アプリケーションが増加している [1].

高機能な情報系のアプリケーションの開発を効率化するためには，高機能な汎用OS（情報系OS）の利用が不可欠である．しかしながら，汎用OSはそれ自身の規模が大きいため，完全な検証が事実上不可能である [2]．例えばWindowsやLinux等の汎用OSのセキュリティホールは収束することなく，次々と発見されている．また，実行時間の予測可能性が悪くリアルタイム性が低い [3]．そのため，制御系アプリケーションを汎用OS上で実現すると，制御系システムと連携する制御系アプリケーションに要求される高い信頼性やリアルタイム性を満たすことができない．

この問題を解決する方法として，高い信頼性とリアルタイム性を実現できるRTOS（制御系OS）と情報系OSを単一のシステム（シングルプロセッサ）上で並列に動作させ，制御系アプリケーションは制御系OS上で，情報系アプリケーションは情報系OS上でそれぞれ実行する方法が考えられる．

制御系OSと情報系OSを単一のシステムで動作させる手法としては，ハイブリッドOS方式 [4, 5, 6] がある．本方式では，図1に示すように，各OSは

システムの全てのリソースに対してアクセスが可能な特権レベルで動作するため，制御系 OS の持つ高い信頼性とリアルタイム性が情報系 OS により脅かされる可能性がある．具体的には，情報系 OS に不具合等があり誤動作した場合，制御系 OS の用いるメモリやデバイスに危害を及ぼしたり，全ての割込みを禁止してプロセッサを占有してしまう可能性がある．

VM 方式 [7, 8] は，Virtual Machine Monitor（VMM）が並列実行する各 OS（ゲスト OS）に対して，プロセッサやデバイスを仮想化した Virtual Machine（VM）を提供することで，複数の OS を並列に動作させる．ゲスト OS は，VMM より低い特権レベルで動作し，デバイスへの直接的なアクセスや特権命令の実行が禁止されている．そのため，情報系 OS をゲスト OS として動作させ，制御系 OS を VMM と同じ特権レベルで動作させれば，制御系 OS の信頼性を確保できる．本方式を効率的に実現するためには，ハードウエア機構によるサポート（VM 支援機能）が必須であり [9]，汎用 PC やサーバー向けのプロセッサに導入が進んでいる．

しかしながら，組込み向けのプロセッサにおいて，VM 支援機能を持つ一般的に入手可能なプロセッサは現状存在しない．VM 支援機能に近い機能として，組込みシステムで広く用いられている ARM プロセッサが持つセキュリティのためのハードウェア機構である TrustZone[10] が挙げられる．

本研究では，TrustZone を用いて制御系 OS と情報系 OS で構成されるハイブリッド OS 方式の高信頼化を実現する．具体的には，情報系を制御系より低い特権レベルで動作させることで，制御系のリアルタイム性を保つ一方で，情報系で発生した不具合の制御系への波及や，情報系によるプロセッサの占有を防止する．

TrustZone を利用して，本手法を実現するソフトウェアモジュールを Safety Gate （SafeG）と呼ぶ．SafeG を用いたシステムの概要図を図 2 に示す．

本論文では，まず 2 章で車載情報システムが満たすべき要件について述べる．3 章で本研究で使用した TrustZone の機能について述べる．4 章で SafeG の設計について述べる．5 章で SafeG の性能を評価する．6 章で関連研究と SafeG の比較した結果について述べる．7 章で本論文のまとめを行う．

2 車載情報システムの要件

本研究で対象とする車載情報システムの要件は，次のように定義される．

(**a**) 制御系 OS と情報系 OS がそれぞれ一つずつ，シングルプロセッサで動作すること．

(**b**) デバイスは，各 OS で占有する占有デバイスと共有する共有デバイスとに分けられる．共有デバイスの操作は，情報系 OS から制御系 OS に操作を依頼することで実現する．

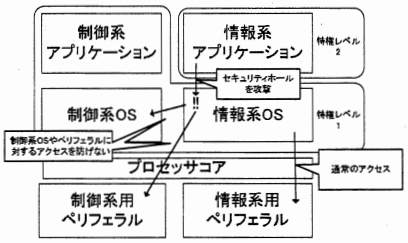

図 1: ハイブリッド OS 方式

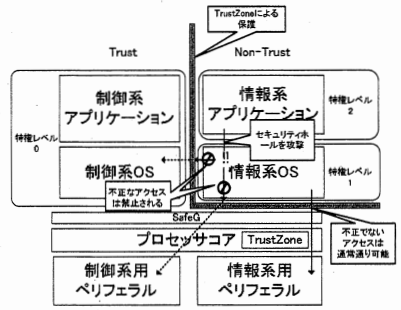

図 2: SafeG によるハイブリッド OS 方式

(**c**) 制御系を情報系よりも優先して実行すること．

(**d**) 制御系の割込みの最悪応答時間が一意に定まること．

(**e**) 実行オーバヘッドをできる限り小さくすること．

(**f**) 制御系が情報系から保護されること．

(**g**) 情報系 OS の書き換えを最小限にすること．

(**h**) SafeG は検証可能な程度に小さくすること．

(**i**) 情報系の確実な監視と停止ができること．

（a）（b）は機能に対する要件である．車載情報システムでは，多くの OS が並列に動作する必要はなく，1 つの制御系 OS と 1 つの情報系 OS の計 2 つの OS が並列に実行できればよい．共有デバイスに関する操作の実現方法については，本論文では扱わず具体的な実現方法については今後の課題とする．

（c）（d）は，リアルタイム性に関する要件である．情報系及び SafeG の実行により，制御系のリアルタイム性に影響を与えないよう定めた．(d) に関しては，制御系の割込み応答時間の最悪値の上限が，情報系の割込み禁止区間の長さに依存せず定まることが重要であるため定めた．

（e）は，性能に関する要件である．SafeG の導入によって発生する OS の切り替えや，割込み処理に伴うオーバヘッドをできる限り小さくする必要がある．

(f) (g) (h) (i) は，信頼性に関する要件である．(f) は，情報系 OS に不具合が発生した場合にも，その影響が制御系に波及しないように保護するため定めた．本手法により，制御系 OS と情報系 OS を同時にシングルプロセッサで動作させるには，各 OS の変更は避けられない．ソフトウェアの検証は，規模と変更が大きくなるほど困難となる．規模の大きな情報系 OS には，可能な限り変更を加えないようにするべきであるため，(g) を定めた．SafeG はシステム全体の信頼性の要であり，それ自身の信頼性が低下するとシステム全体の信頼性が低下するため (h) の要件を定めた．(i) は，情報系システムの信頼性を高めるために定めた．

3 TrustZone

本章では，ARM プロセッサの TrustZone について説明する．本研究において TrustZone を用いた理由については，4 章で述べる．

3.1 Trust と Non-Trust

ARM プロセッサは，2 段階の特権レベルを持つ．特権レベル 1（Privileged Mode）はシステムの全てのリソースに対してアクセス可能であり，特権レベル 2（User Mode）は，リソースに対するアクセスが制限されている．汎用 OS では，特権レベル 1 で OS を，特権レベル 2 でアプリケーションを実行する．TrustZone は，この特権レベルに直交するプロセッサの状態として，Trust 状態と Non-Trust 状態を追加する．

Trust 状態では既存の特権レベル 1，2 と同様の振る舞いが可能である．一方，Non-Trust 状態はプロセッサの特権レベルが 1 であっても，Trust 状態からのみアクセス可能と設定されたメモリ空間（メモリやペリフェラル）にはアクセスすることができず，一部のプロセッサに対するクリティカルな操作も制限される．

また，TrustZone を制御する目的で，Secure Monitor モードと呼ばれるモードがプロセッサに追加されている．Trust 状態と Non-Trust 状態の切り替えはこの Secure Monitor モードを経由して行う．

Trust 状態と Non-Trust 状態では，MMU 関連のレジスタなどの一部の制御レジスタがバンクレジスタとなっており，プロセッサの Trust/Non-Trust の状態によって自動的に切り換えられる．これらのレジスタは，Trust/Non-Trust 状態の切り替えの時に保存復帰する必要がない．一方，汎用レジスタはバンクレジスタになっておらず，Trust/Non-Trust 状態の切替え時に保存と復帰が必要となる．

3.2 アドレス空間の分割

TrustZone では，アドレス空間を Trust 状態でのみアクセス可能な領域（Trust 領域）と，両状態からアクセス可能な領域（Non-Trust 領域）とに分割する．

プロセッサコアは，バスアクセス時に現在の状態（Trust/Non-Trust）を示す信号をバスへ出力する．バスコントローラやペリフェラルは，この信号を用いてどちらの状態からアクセスされているか判断する．バスコントローラで判断する場合は，スレーブ単位で領域の設定を行う．メモリに関しては，ペリフェラルと同様にバスコントローラでスレーブ単位で設定する方法と，TrustZone に対応したメモリコントローラを用いて，一つのメモリ内を各領域に設定する方法がある．

3.3 割り込み

ARM プロセッサには，FIQ と IRQ の 2 種類の割込み信号入力が存在し，OS は通常 IRQ のみを用いる．FIQ と IRQ が発生すると，はそれぞれ異なる割り込みベクタが実行される．ベクタ実行時の振る舞い（レジスタ等の保存状態）が異なるため，それぞれのベクタに配置するハンドラには互換性がない．TrustZone では，Trust と Non-Trust 状態でそれぞれ独立した割込みベクタを持つ（FIQ・IRQ 共に）．Trust 状態で割込みが発生すると Trust 側の割込みベクタを，Non-Trust 状態で割込みが発生すると Non-Trust 側の割込みベクタを実行する．また，Secure Monitor モードも割込みベクタを持ち，FIQ/IRQ が発生した場合に，前述のように各状態の割込みベクタを実行するか， Secure Monitor モードに遷移して Secure Monitor モードの割込みベクタを実行するか選択可能である．

ARM プロセッサでは，FIQ と IRQ をそれぞれ独立に禁止・許可することが可能である．TrustZone では，Non-Trust 状態で FIQ を禁止できないように制限することが可能である（IRQ は禁止可能である）．Non-Trust 状態で扱うデバイスでは IRQ を用い，Trust 側で扱うデバイスでは FIQ を用いることにより，Non-Trust 状態時に Trust 側の割込みが禁止されないようにすることが可能である．

4 SafeG の設計

本章では，TrustZone を用いた理由と，SafeG の設計について述べる．

2 種類の異なる OS をシングルプロセッサ上で並列に実行する場合に考慮しなければいけない要素は，アドレス空間の分離，割込みハンドリング，OS 間のスケジューリング，デバイスの共有である．

4.1 TrustZone の使用

制御系の信頼性を確保するためには，制御系が用いるメモリやペリフェラルといったリソースを情報系からアクセスできないよう保護すればよい．情報系からのアクセスを制約するには，情報系の動作時のプロセッサの特権レベルを制御系のリソースに対するアクセスに必要な特権レベルより低く設定すればよい．

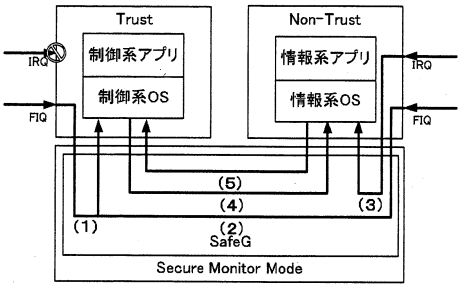

図 3: 構成と実行パス

特権レベルを下げる方法としては，ソフトウェアのみによる方法と，ハードウェアによる支援機能（VM 支援機能）を用いた方法がある．ソフトウェアのみによる方法を低実行オーバヘッドで実現するためには，OS を大きく改変する必要がある．[9] また，プロセッサのアーキテクチャによっては，完全な保護が実現困難な場合もある [11]．一方，VM 支援機能はプロセッサに対して新たな特権レベルを導入する．VM 支援機能を用いることで，OS の改変を必要とせずに完全な保護を低実行オーバヘッドで実現可能とする．

VM 支援機能としては，x86 における Intel の VT-x や AMD の AMD-V があり，PowerPC では HyperVisor モードがある．これらのプロセッサは汎用 PC やサーバ用途として設計されているため，消費電力が大きすぎるなどの問題があり車載情報システム向けとして利用できない．

一方，TrustZone は，組込みシステムで広く普及している ARM プロセッサでサポートされているセキュリティのためのハードウェア機構である．しかしながら，3.1 節で述べた特徴より，本研究で実現したい制御系の保護に有用である．具体的には，プロセッサが Non-Trust 状態の場合に Trust 領域に配置されたリソースへのアクセスが禁止される機能は，リソースの保護に対して有効である．また，Non-Trust 状態で FIQ 割込みを禁止できない機構は，制御系のリアルタイム性の確保に有効である．そのため，本研究では TrustZone を用いて，制御系の保護を実現することにした．

4.2 システムの全体構成

システム全体構成を図 3 に示す．制御系は Trust 状態で，情報系は Non-Trust 状態で動作させる．SafeG は，プロセッサ状態の変更や割込みの分配といった TrustZone の制御を行うため，Secure Monitor モードで動作させる．

制御系が用いるリソース（メモリとペリフェラル）は，Trust 領域とし，それ以外のリソースは Non-Trust 領域とする．これにより情報系からは制御系のリソースへのアクセスは禁止されるため，要件（f）を満たすことになる．情報系が，Trust 領域に設定されたリソースにアクセスを試みると，例外が発生して SafeG が呼び出される．

表 1: 実行優先度

実行優先度	処理内容
高	制御系割込み処理
	制御系通常処理
	情報系割込み処理
低	情報系通常処理

4.3 割込みハンドリング

3.3 節で述べたように，TrustZone では，Trust 状態で FIQ を Non-Trust 状態で IRQ を使用することを前提としている．この方針に従い，SafeG では制御系 OS で FIQ を情報系 OS で IRQ を用いることにした．

さらに，TrustZone の機能により情報系を実行中（Non-Trust 状態）は，FIQ を禁止できないように設定する．これにより，制御系の割込みの最悪応答時間が一意に定まるようになり，要件（d）を満たすことが出来る．また，情報系が割込みを禁止した状態で暴走したとしても，制御系側のタイマやウォッチドッグタイマの割込みより制御系が実行される．すなわち，制御系による情報系の確実な監視と停止が実現できるため，要件（i）を満たす．

各割込みの発生時は，それぞれの OS の割込みベクタではなく，Secure Monitor モードで動作する SafeG の割込みベクタを実行するよう設定する．全ての割込みをいったん SafeG で受付けることにより，SafeG で割込みの発生頻度や間隔を監視することが可能となる．

4.4 実行優先度の設計

要件（c）より，制御系は情報系より優先して動作する必要があるため，処理単位毎の実行優先度を表 1 に示すように定めた．

前述の通り，情報系実行中は FIQ を禁止できないように設定し，制御系では FIQ を使用するため，制御系の割込み処理は情報系より優先して実行される．さらに，制御系の実行時は常に IRQ を禁止することで，制御系の通常処理も情報系よりも優先して動作する．

情報系を実行するタイミングは，制御系に実行するべき処理が無くなった場合であり，SafeG を呼び出すことにより実現する．

4.5 実行パス

前述の割込みのハンドリングや実行優先度の設計に従い SafeG を実装すると，割込みハンドリングと OS の切り替えに関して，図 3 に示す 5 つの実行パスができる．各パスは，全ての割込みを禁止した状態で実行される．

実行パス（1）はプロセッサが制御系（Trust 状態）を実行中に FIQ が発生した場合のパスである．FIQ

が入力された時点で行っていた処理を中断し，SafeGのFIQベクタに置かれた割込みハンドラを実行する．割込みハンドラでは，必要に応じてFIQ発生毎の処理（例えば割込み発生頻度の監視）を行った後，制御系OSのIRQベクタに置かれた割込みハンドラを実行する．この際，制御系OSがSafeGを経由せずにIRQを受けつけた場合と同じ状態になるようにレジスタの値を調整する．そのため，制御系OSの割込みハンドラはSafeG導入によって改変する必要はない．

実行パス（2）はプロセッサが情報系（Non-Trust状態）を実行中にFIQが発生した場合のパスである．実行パス（1）と同様に最終的には制御系OSの割込みハンドラを実行するが，OSの切り替え処理が必要である．具体的には，情報系OSのコンテキストとして汎用レジスタ等の32bitレジスタを38個保存した後，制御系OSのコンテキストとして同数のレジスタを復帰する．

実行パス（3）はプロセッサが情報系（Non-Trust状態）の処理を実行中にIRQが発生した場合のパスである．IRQが入力された時点で行っていた処理を中断し，SafeGのIRQベクタに置かれた割込みハンドラを実行する．割込みハンドラでは，必要に応じてIRQ発生毎の処理（例えば割込み発生頻度の監視）を行った後，情報系OSのIRQベクタに置かれた割込みハンドラを実行する．実行パス（1）と同様に，情報系OSの割込みハンドラは，SafeG導入前と同じ状態になるようレジスタの値を調整するため，情報系OSの割込みハンドラを改変する必要はない．

実行パス（4），(5）は制御系（Trust状態）と情報系（Non-Trust状態）を切り換えるパスである．実行パス（4）は，制御系で実行すべき処理が存在しない場合に，制御系OSからSafeGを呼び出すことにより実行されるパスである．SafeGの呼び出しは，トラップ命令の一種であるSMC命令により実現する．呼び出し後，制御系のコンテキストを保存して，情報系のコンテキストを復帰して，情報系の中断元にリターンする．実行パス（5）は，デバイス共有の目的で使用することを想定しており，情報系から制御系に移る．

5 評価

本章では，SafeGが要件（e）（d）（h）（g）を満たしているかどうか，実機を用いて評価する．

評価環境としては，TrustZoneをサポートしているARM1176jzfプロセッサを搭載したPB1176JZF-Sボードを用いた．コアクロックは210MHz，キャッシュは命令・データ共に32KBである．制御系OSとしては，TOPPERS/ASPカーネル1.3.1（ASP）を情報系OSとしては，Linux 2.6.24を用いた．実行時間の測定に関しては，測定毎にキャッシュをパージする条件で行った．

表2: SafeGの実行時間

パス	実行時間
制御系OS実行時にFIQ入力（実行パス（1））	0.7μs
情報系OS実行時にFIQ入力（実行パス（2））	1.6μs
情報系OS実行時にIRQ入力（実行パス（3））	1.2μs
制御系OSから情報系OSに移行（実行パス（4））	1.5μs
情報系OSから制御系OSに移行（実行パス（5））	1.7μs
ASPの割込みベクタから割込み禁止解除まで	5.1μs

5.1 オーバヘッド

要件（e）を満たしているか評価する．SafeGの導入により，オーバヘッドが発生する箇所は，4.5節で述べた割込み入口処理とTrust/Non-Trustの切り替えのための実行パスであり，通常のデバイスやメモリのアクセスに関するオーバヘッドは発生しない．

それぞれの実行パスの実行時間の計測結果と，比較評価のためのASPの割込みベクタから割込み禁止の解除までの実行時間を表2に示す．

各実行パスの実行時間をASPの割込みベクタから割込み禁止が解除されるまでの時間と比較すると，SafeGのオーバヘッドは十分に小さいと言えるため，要件（e）を満たしていると言える．

実行パス（2）（4）（5）は，Trust/Non-Trustの切り替え処理（コンテキストの保存・復帰）を行うため，実行パス（1）（3）と比較して実行時間が大きくなっている．現状では，実行するOSの種類に前提を置いていないため，プロセッサの持つ全レジスタ（38個）を保存・復帰している．なお，特定のOSを前提に最適化を行うと，保存・復帰するレジスタの数を減らすことが可能である．

5.2 制御系のリアルタイム性

要件（d）を満たしているかを確認するために，SafeGを導入することによって，制御系の割込み応答時間がどのように変化するかを測定した．測定は，情報系でXウィンドウシステムを起動した状態で，制御系が用いる周期タイマの割込みが発生してから，OSの割込みハンドラを経由してユーザーの割込みハンドラが実行されるまでの時間を測定した．比較対象として，ASPをSafeGなしで動作させた場合（ASPのみ）も測定した．それぞれの条件で100000回測定した結果を図4に示す．

両者の最も頻度が高い箇所を比較すると，SafeGを導入することにより，1.4μs応答時間が増加している．これは，割込みをSafeGが一旦受付けることによる（図3の実行パス（2））増加である．前節の測定結果より，実行パス（2）の実行時間は1.6μsであり，ほぼ一致している．

また，SafeG導入時は，最も頻度が高い箇所から最大1.5μ程度のジッターが発生している．ジッターの発生原因としては，2種類の原因が考えられる．一つ目は，SafeGの情報系の割込みの実行パス（図3の実行パス（3））の実行中に制御系の割込み（今回の測定では周期タイマ割込み）が発生することによるものである．SafeG実行中は全ての割込みを禁止

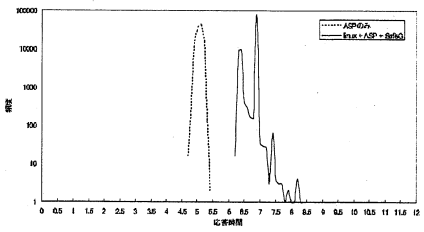

図 4: SafeG による ASP の割込み応答遅延

表 3: コード・データサイズ (バイト)

	text	data	bss	全体
SafeG	1520	0	448	1968
ASP	34796	0	83140	117936
Linux	1092652	148336	89308	1330296

して実行さるため，実行パス（3）を実行中に制御系の割込みが発生しても，情報系 OS に制御が移るまで制御系の割込みは受付けられない．二つ目は，キャッシュによる影響と考えられる．制御系と情報系でキャッシュを共有しているため，情報系の振る舞いによって，制御系実行時のキャッシュの振る舞いが変わり，ジッターが発生すると考えられる．

評価の結果，SafeG を導入することで，最悪値およびジッターは増加しているものの，収束しているため，要件（d）を満たしていると言える．

5.3 コードサイズ

要件（h）を満たしていることを確認するために SafeG のコードサイズについて評価する．

SafeG と ASP と Linux それぞれのコードとデータのサイズを表 3 に示す．SafeG は全体で 1968 バイトとなっている．ASP と比較しておよそ 1/60 程度のサイズであり，十分に検証可能であると考えられるため，要件（h）を満たしているといえる．なお，SafeG の bss セクションのうち，304 バイトは，OS 切り替えの際の各 OS のコンテキストを保存するためのの領域である．

5.4 情報系 OS の変更項目

SafeG では要件（g）より，特に情報系 OS の書き換えが最小になるように設計した．その結果，情報系 OS の書き換えが必要となる点は以下の通りである．

1. 通常のベクタを用いる場合は，ベクタテーブルのアドレスをベクタベースアドレスレジスタに設定する．
2. 制御系の利用するメモリやデバイスをアクセスしないように変更する．
3. 情報系が本来利用していたデバイスを制御系と共有することになった場合，SafeG へ依頼してアクセスするように書き換える．

（1）に関しては，今回の情報系 OS として利用した Linux の場合，ハイベクタと呼ばれる仕組みを利用する設定となっているため該当しない．(2)（3）は共にデバイスドライバに関する設定であり，カーネル本体に変更を加える必要はない．よって，情報系 OS に手を加える箇所は存在しなかったため，要件（g）を満たしていると言える．

6 関連研究

SafeG は VM 支援機能を用いたハイブリッド OS 方式と位置づけることができる．そこで，VM 方式やハイブリッド OS 方式との比較を行う．

VM 方式は並列実行する各 OS に対して，プロセッサやデバイスを仮想化した VM を提供する方式であり，各 OS はお互いの存在を意識しない．プロセッサが VM 支援機能を持たない場合にはソフトウェアにより仮想化を実現する．ハイブリッド OS 方式は，OS 自身を変更することにより，2 種類の OS をシングルプロセッサで動作させる．

6.1 サポートする OS の数

VM 方式は計算機の利用効率の向上を目指しており，できるだけ多くの OS を同一の CPU 上で動作させることを目的としている．一方，SafeG やハイブリッド OS 方式では，2 種類の OS が動作すればよい．

6.2 リアルタイム性

VM 方式はサーバー用途の目的で用いられることが多いため，リアルタイム性より，スループットを優先して設計されている．例えば，Denali[8] や Xen[7] などでは，割込み処理を非同期に行うことでシステム全体としてのスループットの向上を実現している．

SafeG やハイブリッド OS 方式は，制御系 OS のリアルタイム性の確保を目的に設計されている．リアルタイム性が要求されるデバイスは制御系 OS で占有し，操作や割込みは直接扱う．

6.3 信頼性

VM 方式はゲスト OS の特権レベルを VMM より下げ，特権命令の実行やデバイスへの直接的なアクセスを禁じることで実現しているため，ゲスト OS の独立性が高く，ゲスト OS 間で不具合が伝播しない．ハイブリッド OS 方式では，各 OS は全てのリソースに対してアクセスが可能な特権レベルで動作する．各 OS はデバイスへの直接的なアクセスや特権命令の実行が可能であるため，OS 間の保護が実現できない．

SafeG では，VM 方式と同様に情報系 OS の特権レベルを制御系 OS より下げることで，情報系の不具合が制御系に波及しないようにしている．さらに，VM 支援機能を用いることにより，確実な保護を低い実行オーバヘッドで実現している．

6.4 ゲストOSの変更量

VM方式には，ゲストOSの変更量の観点から，full-virtualizationとpara-virtualizationの2種類に分類できる．full-virtualizationはゲストOSの改変を必要としない方法で，プロセッサがVM支援機能を持たない場合には，実行オーバヘッドが大きい．para-virtualizationは，ゲストOSを仮想マシン向けに改変する方法である．ハイブリッドOS方式は，各OSの変更は必要不可欠となる．

SafeGは，VM支援機能を用いることで，情報系OSの改変を最小とする．TrustZoneを用いた手法では，情報系OSの改変は必要なかった．

7 終わりに

本論文では，制御系と情報系により構成されるシステムの高信頼化を実現するため，車載情報システムに必要な要件を挙げ，セキュリティ支援ハードウェアであるTrustZoneを用いたソフトウェアモジュールであるSafeGの設計について述べた．また，SafeGが要件を満たしているか評価を行った．

SafeGの設計は，要件である制御系が常に情報系よりも優先して動作すること，制御系が情報系から保護されること，情報系の確実な監視と停止ができることを満たすように行った．

また，実機を用いて，制御系の割込み最悪応答時間が一意に定まるか，小さな実行オーバヘッドで実行可能か，情報系OSの書き換え量，SafeG自身の実装が検証可能な程度に小さくすることが可能であるか評価した．

その結果，最悪応答時間は悪化するものの最悪値は抑えることができることを確認した．割込み入り口処理におけるオーバヘッドおよびOS切り替えのオーバヘッドに関しては，低く抑えることができることを確認した．また，特定のOS向けに最適化を行うことで高速化が可能である．SafeG自身の実装もRTOSと比較しても小さく，十分検証可能なサイズであることを確認した．

今後の課題としては，まずマルチコアの対応が挙げられる．現状のSafeGはシングルプロセッサを対象としてるが，マルチプロセッサを導入して，制御系と情報系を異なるコアで動作させた場合の制御系の保護の方法について研究する．次に共有デバイス操作の実現の検討が挙げられる．情報系OSから制御系OSにデバイス操作を依頼することで実現する予定である．

謝辞

本研究を進めるに当たり，貴重なご意見をいただいたトヨタ自動車株式会社BR制御ソフトウェア開発室の方々をはじめ，ご協力くださった名古屋大学大学院情報科学研究科附属組込みシステム研究センター車載マルチメディアシステム向けOSプロジェクトのメンバー各位に厚く御礼申し上げます．

参考文献

[1] 奥出真理子，遠藤芳則，中村浩三，上脇　正，齊藤雅彦，川股幸博，友部　修，杉浦一正：ITSにおける車載情報システムの検討，情報処理学会論文誌，Vol.42, No.7, pp.1736-1743 (2001)

[2] Kinebuchi, Y., Koshimae, H., Oikawa, S. and Nakajima, T.: Virtualization Techniques for Embedded Systems, RTCSA 2006, (2006)

[3] 石綿陽一，松井俊浩：汎用OSとデバイスドライバを共有できる実時間オペレーティングシステム，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.97, No.569, pp.41-48 (1998)

[4] 保田信長，飯山真一，冨山宏之，高田広章，中島　浩：LinuxとITRONによるハイブリッドOSの設計と実装(特集実時間処理, 組込システム及び一般)，情報処理学会研究報告. SLDM, [システムLSI設計技術]，No.33, pp.45-50 (2004)

[5] Takada, H., Iiyama, S., Kindaichi, T. and Hachiya, S.: Linux on ITRON: A Hybrid Oper-ating System Architecture for Embedded Systems, saint-w, (2002)

[6] 新井利明，関口知己，佐藤雅英，木村信二，大島　訓，吉澤康文：汎用OSと専用OSを高効率に相互補完するナノカーネルの提案と実現(システムソフトウェア設計・構成論)，情報処理学会論文誌，Vol.46, No.10, pp.2492-2504 (2005)

[7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Wareld, A.: Xen and the art of vir-tualization, SIGOPS Oper. Syst. Rev., Vol.37, No.5, pp.164-177 (2003)

[8] Whitaker, A., Shaw, M. and Gribble, S. D.:Scale and performance in the Denali isola-tion kernel, SIGOPS Oper. Syst. Rev., Vol.36, No.SI, pp.195-209 (2002)

[9] Hand, S., Wareld, A. and Fraser, K.: hard-ware virtualization with Xen, ;login: The USENIX Magazine, Vol. 32, No. 1, pp. 21-27 (2007)

[10] Alves, T. and Felton, D.: TrustZone: Inte-grated Hardware and Software Security, ARM (2004)

[11] Popek, G.J. and Goldberg, R.P.: Formal requirements for virtualizable third generation architectures, SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles, New York, NY, USA, ACM, p.121 (1973)