論　文

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†

Enhancing Reliability in Hybrid OS System with Security Hardware*

Kenichiro NAKAJIMA†, Shinya HONDA†, Shigeharu TESHIMA†, and Hiroaki TAKADA†

あらまし　近年，カーナビゲーションシステムの高機能化は著しく，一台のコンピュータ上で，ナビゲーション等の情報系の機能に加えて，ブレーキやエンジン等と連携する制御系の機能も実行する必要がある．制御系の機能は情報系の機能と比較して，高い信頼性やリアルタイム性が要求される．そのため，汎用 OS と RTOS をハイブリッド OS 方式や VMM 方式を用いて，並列に実行し，それぞれの OS で情報系の機能と制御系の機能を実行する方法が考えられる．しかしながら，ハイブリッド OS 方式を用いて OS を並列に実行した場合，制御系を情報系から保護できず，システム全体としての信頼性やリアルタイム性が確保できない．また，VMM 方式では実行オーバヘッドやリアルタイム性が問題となる．本論文では，組込み向けプロセッサがも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 方式を高信頼化する方法を提案し，実機による評価を行った．本提案方式を用いることで，制御系を情報系から保護でき，システム全体としての信頼性やリアルタイム性が確保できる．評価の結果，実行オーバヘッドは十分小さく，また，情報系の実行状態にかかわらず，制御系のリアルタイム性が確保できることが分かった．本提案方式は，実行する OS に対する制約が緩やかであるため，携帯電話，NC 工作機械などの，制御系と情報系から構成される組込みシステムに適用可能である．

キーワード　ハイブリッド OS，信頼性，セキュリティ，リアルタイム OS

1. まえがき

近年，カーナビゲーション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もつ機種もある．更に，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，制御系のアプリケーションが増加している [1]．制御系のアプリケーションは，制御系システムと連携するため，情報系アプリケーションと比較して，高いリアルタイム性と信頼性が要求される．

高機能な情報系のアプリケーション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，情報系 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 等の情報系 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情報系 OS 上で実現すると，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リアルタイム OS（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．

2 種類以上の OS をシングルプロセッサで動作させる

† 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，名古屋市
Center for Embedded Computing System, Nagoya University, Nagoya-shi, 464–8601 Japan
* 本論文はシステム開発論文である．

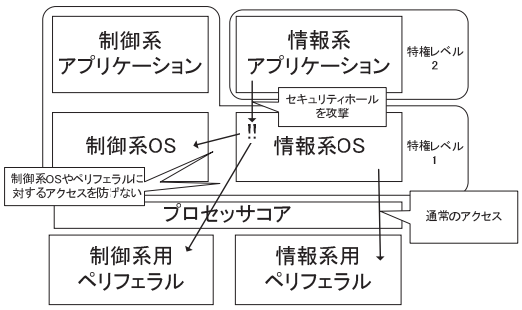


図 1　ハイブリッド OS 方式
Fig. 1　Hybyrid OS system.

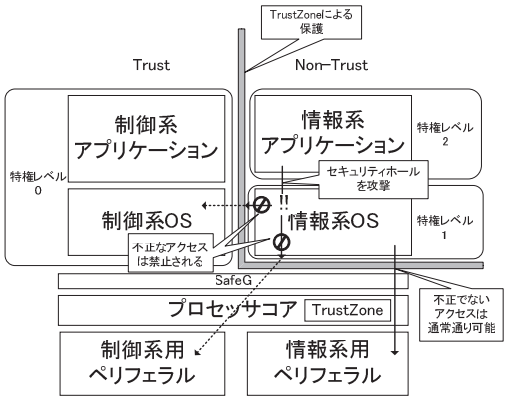


図 2　SafeG によるハイブリッド OS 方式
Fig. 2　Hybyrid OS system with SafeG.

手法としては，ハイブリッド OS 方式 [4]〜[6]，VMM 方式 [7], [8]，追加ハードウェアによる方式 [9] がある．各方法には信頼性と実行性能において長所と短所があり，適用するシステムごとに適切な方法を選択する必要がある．ハイブリッド OS 方式では，図 1 に示すように，各 OS が直接ハードウェアを操作できるため，単独で実行した場合に近い実行性能を得られるという長所が存在する．しかし，各 OS 間の特権レベルが同一であるために OS 間の独立性を保つことが難しい．そのため，信頼性の低い情報系 OS のセキュリティホールが攻撃され，不具合が発生した場合，その問題が制御系 OS にも波及してしまうという問題が発生する．例えば，情報系 OS が割込みを禁止した状態で暴走すると，制御系 OS の動作や割込みの受付けができなくなり，制御系 OS のリアルタイム性が確保できなくなる．また，情報系 OS が制御系 OS の用いるメモリを不正にアクセスすると，制御系 OS を破壊することができる．

そこで本研究では，セキュリティ支援ハードウェアである TrustZone [10] を用いてハイブリッド OS 方式における OS 間の独立性を高めることで，ハイブリッド OS 方式の高信頼化を実現する．TrustZone を制御して，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．本手法は，TrustZone の機能を用いて，情報系 OS を制御系 OS よりも実行優先度を低くすることで，制御系 OS のリアルタイム性を保ち，情報系 OS によるプロセッサの占有を防止する．更に，情報系 OS を制御系 OS より低い特権レベルで動作させることで，情報系 OS から制御系 OS の用いるメモリへの不正アクセスを防ぐ．

本論文では，まず **2.** でカーナビゲーションシステム用ハイブリッド OS システムが満たすべき要件について述べる．**3.** で本研究で使用した TrustZone の機能について述べる．**4.** で SafeG の設計について述べる．**5.** で SafeG の性能と TrustZone を評価する．**6.** で関連研究について述べる．**7.** で本論文のまとめを行う．

2. カーナビゲーションシステム用ハイブリッド OS システムの要件

本研究で対象とするカーナビゲーションシステム用のハイブリッド OS 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） 両 OS から使用するデバイスを共有する機能をもつこと．

（c） 共有しないデバイスに対する操作は，SafeG の導入により実行オーバヘッドが増加しないこと．

（d） 情報系及び SafeG の実行が，制御系のリアルタイム性に影響を与えないこと．

（e） SafeG の実行時間は制御系の割込み応答時間以下であること．

（f） 情報系に不具合が発生した場合，その問題が制御系に波及しないこと．

（g） SafeG の導入により情報系 OS のコア部分（ディスパッチャや割込みエントリ）の変更が発生しないこと．

（h） SafeG はシステム全体の信頼性の要であるため検証が容易なコードであること．

（i） 情報系に対する安全系を制御系により実現するための機能をもつこと．

（a）でシングルプロセッサを対象とするのは，本研究はセキュリティ支援ハードウェアを用いたハイブリッド OS システム研究の第 1 段階であり，また，マルチプロセッサを利用できないローコストモデルを対象とするためである．マルチプロセッサ対応に関しては今後の課題とする．

（b）（c）はデバイスに関する要件である．カーナビゲーションシステムにおける代表的なデバイスとしては，ディスプレイ，ハードディスク，車載ネットワーク（CAN），タイマが挙げられる．現状，すべてのデバイスは OS 間の共有は必要としていない．ディスプレイは，情報系 OS と制御系 OS にそれぞれフレームバッファを割り当て，制御系のフレームバッファが優先的に表示されるようになっている．ハードディスクは，情報系 OS のみが使用している．CAN は制御系 OS のみが使用している．タイマはそれぞれの OS が独立して保持している．これらの共有しないデバイスに対する操作に関しては，SafeG の導入によって，実行オーバヘッドが増加しないことが望まれる．しかしながら，ハードディスクに関しては，今後，制御系 OS からもデータを書き込みたいケースが考えられる．そこで，本研究では，デバイスを共有するための機構のみを検討し，この仕組みを使って具体的にデバイス共有をする方法については今後の課題とする．

（d）リアルタイム性に関する要件である．制御系の処理は，情報系よりリアルタイム性が高い．一方，経路探索のようにある程度時間を要する処理は，情報系で実行する．そのため，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や SafeG の処理に依存せず定まる必要がある．

（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 の切換や，割込み処理に伴うオーバヘッドは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，情報系の信頼性を制御系と同じ高いレベルに引き上げるのは困難であるため定めた．

（g）に関しては，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べきであるため定めた．一方，制御系 OS は規模が小さいため変更は許容する．

（h）は，SafeG 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定めた．SafeG の検証手段としては，制御パステストとソースコードレビューが挙げられる．

（i）は，情報系に対する安全系を制御系で実現するために定めた．例えば，バックガイドモニタは情報系が実行するが，画像の更新が停止（フリーズ）することは事故につながり危険である．そのため，制御系により情報系の画像の更新を監視し，情報系に不具合が発生して更新が停止した場合は，制御系によりドライバーに通知することで，システムの安全性を高めることができる．

3. TrustZone

本章では，TrustZone のもつ機能について述べる．

3.1 Trust と Non-Trust

ARM プロセッサは，2 段階の特権レベルをもつ．特権レベル 1（Privileged Mode）はシステムのすべ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 状態はプロセッサが特権レベル 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（メモリやデバイス）にはアクセスすることができず，プロセッサに対する一部の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 状態の切換はこの Secure Monitor モードを経由して行う．なお，Secure Monitor モード実行中に割込みを受け付けるとレジスタを破壊してしまうため，原則割込みを禁止して実行する必要がある．

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 状態の切換のとき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 状態の切換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デバイスは，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，デバイス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 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 は発生すると，それぞれ異なる割込みベクタが実行される．ベクタ実行時の振舞い（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もつ（FIQ・IRQ ともに）．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もち，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

本章は，SafeG の設計として，**2.** で述べた要件のうち SafeG の設計により達成可能な項目である，(a) (b) (c) (d) (f) (g) (i) の実現方法について述べる．

4.1 2種類の OS のシングルプロセッサ上での実行

システム全体構成を図 3 に示す．要件（a）を実現するため，TrustZone を用いて，情報系 OS を Non-Trust 側，制御系 OS を Trust 側で動作させ，SafeG によりプロセッサ状態の変更や割込みの分配といった TrustZone の制御を行うことにより，2 種類の OS のシングルプロセッサ上での実行する．なお，SafeG は，TrustZone の制御を行うため，Secure Monitor モードで動作させる．

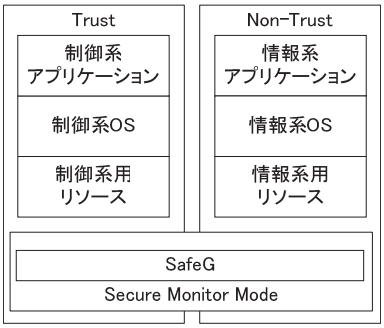


図 3 ハイブリッド OS システムの構成
Fig. 3 System allocation.

表 1 実行優先度
Table 1 Execution priority.

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

3.3 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．

4.2 リソースの配置

要件（c）を実現するため，共有しないデバイスは，それぞれの OS が直接アクセス可能とする．そのため，制御系が用いるリソース（メモリとデバイス）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，SafeG の導入により操作のための実行オーバヘッドが増加しないため，要件（c）を満たす．

4.3 制御系のリアルタイム性の確保

要件（d）より，制御系が情報系の処理に影響を受けないためには，情報系より優先して動作させる必要がある．そのため，処理単位ごとの実行優先度を表 1 に示すように定めた．

この実行優先度を実現するために，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．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のすべての処理よりも優先して実行される．更に，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．制御系は実行するべき処理がなくなった場合に SafeG を呼び出し，情報系に処理を移す．

また，ハードウェア的に情報系を実行中は，制御系の割込み（FIQ）禁止できないようにしているため，制御系の割込み応答時間の最悪値の上限は，情報系の処理（割込み禁止区間も含む）に依存せずに定まる．更に，SafeG の実行パス内にループがないため，SafeG の処理にも依存しない．

4.4 制御系の保護

要件（f）を実現するため，制御系のリソースに対する情報系からのアクセスを禁止する．**4.2** で述べたように，制御系が用いるリソース（メモリとデバイス）は，Trust 領域とし，それ以外のリソースは Non-Trust 領域とする．そのため，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．

また，**4.3** で述べたように，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とにより，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確実に制御系が実行される．

4.5 デバイス共有

要件（b）を実現するため，デバイス共有のための機構を用意する．デバイスの共有の方法としては，SafeG が直接デバイスの制御を行う方法と，いずれかの OS が直接制御を行い，他方の OS は OS 間通信を使って処理を依頼する方法の 2 種類が考えられる．

前者の方法では SafG 内にデバイスドライバを内蔵しなければならず，SafeG の規模が大きくなることや，機器の構成ごとに SafeG の改変が必要となり信頼性が低下する．よって，デバイスの共有については OS 間通信を利用して実現する．更に，ハードディスクの例では，制御系 OS が書き込むデータの方が重要であるため，制御系 OS がデバイスを直接制御し，情報系 OS が OS 間通信により処理を依頼する方法とする．

OS 間通信は，情報系から SafeG を経由して情報系を呼び出す機構と Non-Trust 側のメモリを用いて実現する．具体的には，情報系 OS は依頼したい処理内容とデータを Non-Trust 側のメモリに書き込み，制御系 OS を SMC 命令を使って SafeG 経由で呼び出す．呼び出された制御系 OS は，依頼された処理（デバイスの操作）を行う．SMC 命令を用いた SafeG 経由の呼び出しの詳細については，**4.7** で述べる．

4.6 制御系による情報系に対する安全系実現

要件（i）を実現方法として，制御系による情報系の確実な監視と停止が挙げられる．

監視に関しては，メモリの監視と割込みの監視が考えられる．メモリに関しては，**4.2** で述べたように，情報系のメモリは Non-Trust 領域であるため，制御系からアクセスすることが可能である．

割込みに関しては，TrustZone の機能により，割込み発生時にそれぞれの 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．すべての割込みをいったん SafeG で受け付けることにより，SafeG で割込みの発生頻度や間隔を監視する．この情報を制御系から参照することで割込みの監視を実現する．

停止に関しては，**4.3** で述べたように制御系は情報系より優先して実行されるため，制御系のウォッチドッグタイマの割込み等により，情報系の処理状態に依存せずに情報系を停止（中断）させ，制御を制御系に移すことが可能である．

4.7 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 の切換に関して，図 4 に示す五つの実行パスができる．**3.1** で述べたように，SafeG が動作する Secure Monitor モードでは割込みを受け付けることができないため，各パスは，すべての割込みを禁止した状態で実行される．

4.6 で述べたように，割込みはすべての SafeG で

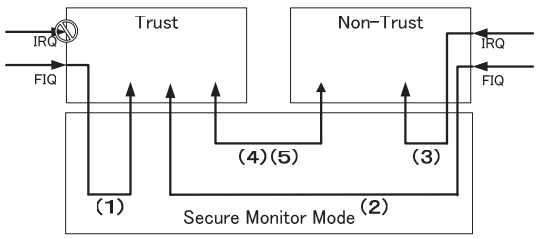


図 4 実行パス
Fig. 4 Execution path.

受け付けるため，SafeG によりそれぞれの OS に割込みを配分する必要がある．その際，要件（g）を実現するため，情報系 OS の割込みエントリを変更する必要がないように，SafeG は，情報系 OS が直接割込みを受け付けた場合と同じ状態になるようにレジスタの値を調整して情報系 OS の割込みエントリを呼び出す．なお，制御系 OS に対しても同様の処理をして呼び出している．

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 発生ごとの処理（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 を受け付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 の切換処理が必要である．具体的には，情報系 OS のコンテクストとして汎用レジスタ等の 32 bit レジスタを 38 個保存した後，制御系 OS のコンテク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 発生ごとの処理（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．呼出し後，制御系のコンテクストを保存，情報系のコンテクストを復帰した後，情報系の中断元に返る．実行パス（5）は，**4.5** で述べた，デバイス共有の目的で使用する．情報系のコンテクストを保存，制御系のコンテクストを復帰した後，制御系 OS のソフトウェア例外ベクタを実行する．

5. 評　　価

本章では，SafeG が要件（d）（e）（h）を満たしているかどうか，実機を用いて行った評価について述べる．また，OS 間の保護の観点による TrustZone の評価について述べる．

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 210 MHz，キャッシュは命令・データともに 32 KB 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．Non-Trust 側が ASP と SafeG に対して影響を与える部分は，キャッシュのみである．よって，実行時間の測定に関しては，測定のループ 1 回ごとにキャッシュをすべてパージする条件で行った．

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，**4.7** で述べた割込み入口処理と Trust/Non-Trust の切換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．

制御系の割込み応答時間としては，“ASP の割込みベクタから割込み禁止解除まで” を測定した．この処理は，OS 内部の処理であり，ユーザが定義した割込みハンドラを呼び出すために以下の処理を行う．これらの処理は，アセンブラで記述されており，ループは存在せず，合計 50 ステップである．なお，5 の後にユーザが定義した割込みハンドラを呼び出す．

1. 割込み発生時のコンテクストを保存する．
（プロセッサモードを 3 回切り換える）
2. 多重割込みか判断し，多重割込みでない場合はスタックを切り換える．
3. 割込み要因を判断し，割込み優先度を設定する．
4. 割込みハンドラのアドレスを取得する．
5. 割込みの禁止解除を行う．

それぞれの実行パスの実行時間の計測結果を表 2 に示す．各実行パスの実行時間は，“ASP の割込みベクタから割込み禁止解除まで” の実行時間より小さいた

め，要件（e）を満たしている．

実行パス（2）（4）（5）は，Trust/Non-Trust の切換処理（コンテクストの保存・復帰）を行うため，実行パス（1）（3）と比較して実行時間がやや大きくなっている．現状では，実行する OS の種類に前提を置いていないため，プロセッサのもつ全レジスタ（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でき，更なるオーバヘッドの削減が可能である．

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，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，制御系のみ実行した場合の制御系の割込み応答時間（ASP），情報系のみ実行した場合の情報系の割込み応答時間（Linux），SafeG を用いて制御系と情報系を並列に実行した場合の制御系の割込み応答時間（SafeG + ASP）と情報系の割込み応答時間（SafeG + Linux）の 4 種類について行った．制御系では 1 ms 周期のシステムティックの更新処理のみを行い，情報系では X ウィンドウシステムを実行し，その上でターミナル，Xeyes，Xclock，top コマンドを実行した状態で計測を行った．測定に用いた割込みは，それぞれの OS のシステムティック用の周期タイマである．割込み応答時間として，タイムアウト割込みが発生してから，各 OS のタイマ割込みハンドラが実行されるまでの時間を 10 万回測定した．測定結果を図 5 に示す．縦軸に発生頻度を対数軸でとり，横軸に応答時間をとっている．

表 2　SafeG の実行時間
Table 2　SafeG execution time.

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

ASP と SafeG + ASP 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，応答時間が 2 μs 増加している．これは，割込みを SafeG がいったん受け付けることによる（図 4 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 1.6 μs であり，ほぼ一致している．

Linux と Linux + SafeG のグラフを比較すると，SafeG を導入することにより，応答時間が 31 μs 増加している．これは，ASP と同様に SafeG が動作するためのオーバヘッドが発生することに加え，制御系が動作していることによってもオーバヘッドが発生するためであると考えられる．Linux の割込み応答時間のグラフに注目すると，制御系と比較して応答性が悪い（グラフには表示していないが，最悪値は 7637 μs である）．これは，Linux 内の割込み禁止区間が長くリアルタイム性がそれほど高くないことを示している．この特性は SafeG を導入しても改善されることはない．一方，SafeG + ASP では応答時間は SafeG の導入によるオーバヘッドにより悪化しているものの，情報系の影響を受けていないことが確認できた．

評価の結果，情報系では割込み応答時間とジッタは増加している．一方，制御系では割込み応答時間の増加は見られたもののジッタは増加しておらず，最悪値の上限が定まるため，要件（d）を満たしていると言える．

5.3 SafeG の検証容易性評価

要件（h）を実現するためには，規模が小さいこと，分岐が少ないこと，割込み禁止で動作することが挙げ

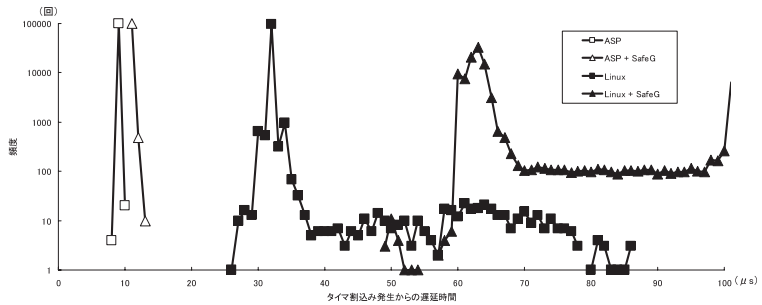


図 5　割込み応答遅延
Fig. 5　Interrupt latency.

表 3 コード・データサイズ（バイト）
Table 3 Code, data size (Byte).

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

られる．規模が小さいと，ソースコードレビューが容易になる．分岐が少ないと，制御パステストの作成数が少なくなる．割込みを許可して動作するプログラムは一般的に検証が困難である．例えば，プログラム中のどの箇所で割込みが入っても正しく動作するか検証する必要がある．

まず，規模に関して評価する．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 バイトとなっており，ASP と比較しておよそ 1/60 程度のサイズであり，十分に検証可能なサイズであると考えられる．なお，SafeG の bss セクションのうち，304 バイトは，OS 切換の際の各 OS のコンテクストを保存するための領域である．

分岐に関しては，SafeG 内の分岐は 4 個あり，このうち 3 個は割込み（IRQ/FIQ）と SMC の受け付け時の OS の判定（制御系か情報系）であり独立している，残りの 1 個は，**5.4.1** で述べるハイベクタの判断であり，IRQ 割込みのみで実施する．そのため，制御パステストを条件網羅（C2）を満たすように作成した場合でも 8 種類に抑えられる．

割込み禁止での動作に関しては，**4.7** で述べたように，TrustZone の規定により，SafeG の実行中はすべての割込みを禁止しており，実現している．

5.4 情報系 OS の変更項目

SafeG は要件（g）より，**4.7** で述べたように，情報系 OS のコア部分の書換えが発生しないように設計した．その結果，情報系 OS の書換えが必要となる点は次の 3 点である．

5.4.1 ベクターテーブル

ARM プロセッサは 0x0 から始まる通常ベクタか，0xffff0000 から始まるハイベクタを選択可能である．SafeG 導入時には SafeG のベクタテーブを通常ベクタに配置する必要があるため，情報系 OS が通常ベクタを使用していた場合は，情報系 OS のベクタテーブルを 0x0 以外のメモリに配置し，その先頭アドレスをベクタベースアドレスレジスタに設定する必要がある．ハイベクタを用いる場合には，SafeG のベクタテーブルと衝突しないため，特に変更は必要としない．

5.4.2 メモリとデバイスの分離

情報系 OS の使用するメモリとデバイスを Non-Trust 領域に配置し，制御系 OS と SafeG の使用するメモリとデバイスを Trust 領域に配置するよう変更する必要がある．情報系 OS の使用するメモリやデバイスの変更は，情報系 OS のハードウェアの構成情報ファイルを変更することにより行う．この変更は通常，情報系 OS を新たなターゲットボードに移植する場合であっても行う必要があり，情報系 OS のコア部分の変更ではない．

5.4.3 デバイスの共有

情報系と制御系で単一のデバイスを共有したい場合は，情報系 OS のデバイスドライバを変更して，SafeG 経由で情報系 OS へ依頼して（SafeG の実行パス（5））アクセスするように書き換える必要がある．なお，具体的なデバイス共有の実現方法については今後の課題とする．

5.4.4 本評価環境における変更項目

本評価環境では，PB1176JZF-S ボードで Linux が動作している環境に対して，SafeG と制御系 OS の導入を行った．ベクタテーブルに関しては，情報系 OS として利用した Linux の場合，ハイベクタを利用する設定となっているため該当しない．メモリとデバイスの分離に関しては，情報系 OS が使用していなかった領域に SafeG と制御系 OS を配置したため，変更する必要がなかった．デバイスの共有に関しては，今回の環境では共有デバイスは存在しなかったため，変更する必要がなかった．

以上により，SafeG の導入により情報系 OS のコア部分の書換えが必要ないため，要件（g）を実現している．

5.5 TrustZone の評価

TrustZone の本来の目的は，セキュリティ支援であるが，ハイブリッド OS 方式の支援機能として用いることにより，**2.** で挙げた要件を満たすハイブリッド OS システムを実現できることが分かった．

更に，以下の 2 点が改善されると，ハイブリッド OS 方式の支援機能としての有用性が増すと考えられる．

1 点目として，Trust 側と Non-Trust 側でキャッシュを分離し，それぞれで管理を行うことができないことである．現時点ではキャッシュが分離されておらず，Non-Trust 側が Trust 側のキャッシュを追い出すことができるため，Trust 側が影響を受けてしまう．

2 点目として，レジスタの保存復帰処理をソフトウェアで行うことになっていることである．Trust 側と Non-Trust 側の切換において，バンクではない 38 個のレジスタの保存復帰処理は常に一定であるため，保存復帰用の命令を用意し，ハードウェアによって保存復帰処理を行うことで，現状よりもパフォーマンスを向上させることができるのではないかと考えられる．

6. 関 連 研 究

OS を並列に実行する方式にはハイブリッド OS 方式，VMM 方式，追加ハードウェアによる方式がある．本章では，これらの方式と本研究で実現した方式を比較する．

ハイブリッド OS 方式は，ハードウェアリソースの利用時に他の OS に影響を与えないように OS を改造することで複数の OS を並列に動作させる方式である．ハイブリッド OS 方式の例として，Linux on ITRON [4], [5] や，DARMA [6] が存在する．

Linux on ITRON では，Linux（ゲスト OS）を ITRON（ホスト OS）上のタスクとすることで，並列に実行するシステムを構築している．この研究では，Linux の割込み処理を ITRON の低優先度のタスクよりも優先して処理することで，Linux の割込み処理の応答性の向上を実現している．

DARMA では，ゲスト OS に，メモリと I/O 装置を振り分け占有させることで，並列に実行するシステムを構築している．ゲスト OS は，割り当てられていないメモリや I/O 装置はハードウェアとして存在しないものとして振る舞う．また，割込みの処理は仮想化されており，OS 間で共有する形となっている．そのため，ゲスト OS の割込み処理を書き換える必要が発生している．

これらのハイブリッド OS 方式では，ホスト OS とゲスト OS はともにシステムのすべ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，実行性能が高めやすいという長所が存在する．しかし，ホスト OS 側のメモリやデバイスがゲスト OS からハードウェア的に保護されておらず OS の独立性がない．そのため，制御系 OS と情報系 OS を並列に実行した場合，制御系 OS のも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．

VMM 方式は，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方式である．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．しかし，VM を管理する必要があるため VMM 自身の実装が複雑となり，VMM 自信の信頼性をどのように確保するかが問題となる．また，ソフトウェアのみで VM を提供する場合にはオーバヘッドが大きくなってしまい実行性能が低下する．VMM 方式を効率的に実現するには，ハードウエア機構によるサポート（VM 支援機能）が必須である [11]．VM 支援機能は汎用 PC やサーバー向けのプロセッサにおいて導入が進んでいる．しかしながら，組込み向けのプロセッサにおいて，VM 支援機能をもつ一般的に入手可能なプロセッサは現状では存在しない．

追加ハードウェアによる方式の一例として，井上らによる VIRTUS [9] が挙げられる．VIRTUS では複数のコアを使うことで，マスター VMM とスレイブ VMM という形で実行状態を分離し，マスター VMM がスレイブ VMM をコントロールすることでセキュリティを確保している．しかしながら，VIRTUS では仮想プロセッサと物理プロセッサのテーブルを作る専用の回路を用意する必要があり，汎用性が低い．VIRTUS は不特定多数の OS を並列に動作させることができる機能をもつ．この機能を利用することで，環境をアプリケーションごとに独立させセキュリティを確保できる．しかしながら，アプリケーションごとに OS を含めた環境を用意するのは，一般的にリソース制約が厳しい組込みシステムでは困難である．また，この方式は，マルチプロセッサを対象としており，本研究で対象としているシングルプロセッサには適用できない．

これらの方式と，本研究で実現した TrustZone と SafeG を用いたハイブリッド OS 方式を比較する．ハイブリッド OS 方式で問題となる制御系の信頼性とリアルタイム性と保護に関しては，本方式では，情報系からは制御系のリソースにアクセスできず，制御系側の割込みも禁止できないことにより保護を実現している．VMM 方式で問題となる規模に関しては，評価で述べたように SafeG の規模は全体で 1968 バイトと，十分検証可能な規模である．また，実行性能も VM 支援機能をもたないプロセッサで実現した VMM 方式

と比較して，本方式は各 OS が直接メモリやデバイスをアクセスできるため実行性能は高い．追加するハードウェアに関しては，本研究では組込みシステムで広く使われている ARM プロセッサの標準機能である TrustZone を利用しているため，汎用性が高い．また，TrustZone を利用したハイブリッド OS 方式は，現時点では本研究以外では開発されていない．

7. むすび

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，カーナビゲーションシステム用のハイブリッド OS 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，定めた用件を SafeG が満たしていること，及び TrustZone の有用性について評価を行った．

提案方式は，カーナビゲーションシステムのみならず制御系と情報系の処理が混在している携帯電話や NC 工作機械などの組込システムに適用可能である．

今後は，更なるオーバヘッドの低減，デバイス共有，マルチコアプロセッサへの対応を行っていく予定である．

謝辞 本研究を進めるにあたり，貴重な御意見を頂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，御協力下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．

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，” 情処学論，vol.42, no.7, pp.1736–1743, 2001.

[2] Y. Kinebuchi, H. Koshimae, S. Oikawa, and T. Nakajima, “Virtualization techniques for embedded systems,” Technical report, RTCSA 2006, Work in Progress Session, Aug. 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，” 信学技報．CPSY，コンピュータシステム，vol.97, no.569, 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，” 信学技報．CPSY，コンピュータシステム，vol.103, no.736, 2004.

[5] H. Takada, S. Iiyama, T. Kindaichi, and S. Hachiya, “Linux on itron: A hybrid operating system architecture for embedded systems,” saint-w, vol.00, p.4, 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），” 情処学論，vol.46, no.10, pp.2492–2504, 2005.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” SIGOPS Oper. Syst. Rev., vol.37, no.5, pp.164–177, 2003.

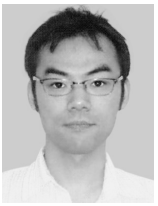
[8] A. Whitaker, M. Shaw, and S.D. Gribble, “Scale and performance in the denali isolation kernel,” SIGOPS Oper. Syst. Rev., vol.36, no.SI, pp.195–209, 2002.

[9] H. Inoue, A. Ikeno, M. Kondo, J. Sakai, and M. Edahiro, “Virtus: A new processor virtualization architecture for security-oriented next-generation mobile terminals,” DAC ’06: Proc. 43rd Annual Conference on Design Automation, pp.484–489, ACM, 2006.

[10] T. Alves and D. Felton, “Trustzone: Integrated hardware and software security,” Information Quarterly, vol.3, pp.18–24, 2004.

[11] S. Hand, A. Warfield, and K. Fraser, “Hardware virtualization with xen,” login: The USENIX Magazine, vol.32, no.1, pp.21–27, Feb. 2007.

（平成 21 年 5 月 22 日受付，9 月 8 日再受付）



中嶋健一郎

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研究員．2003 北陸先端科学技術大学院大学情報科学研究科修士課程了．2006 より現職．

本田　晋也　（正員）

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助教．2002 豊橋技術科学大学大学院情報工学専攻修士課程了．2005 同大学院電子・情報工学専攻博士課程了．2005 名古屋大学情報連携基盤センター名古屋大学組込みソフトウェア技術者人材養成プログラム産学官連携研究員．2006 から現職．リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事．博士（工学）．2002 年度情報処理学会論文賞受賞．情報処理学会各会員．

手嶋　茂晴

京都大学大学院工学研究科情報工学専攻修士課程，名古屋大学大学院工学研究科情報工学専攻博士後期課程了．博士（工学），1986（株）豊田中央研究所入社，現在に至る．2006〜2009 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特任教授/ディレクターなど歴任．

高田　広章　（正員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 東京大学大学院理学系研究科情報科学専攻修士課程了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 より現職．2006 より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長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，情報処理学会，日本ソフトウェア科学会各会員．