Vol.2017-ARC-225 No.33
Vol.2017-SLDM-179 No.33
Vol.2017-EMB-44 No.33
2017/3/10

ARM TrustZone for ARMv8-Mを利用した軽量メモリ保護RTOS

河田 智明[1]　本田 晋也[1]

概要：本研究では ARMv8-M で追加された TrustZone for ARMv8-M を用いて，メモリ保護 OS を低オーバヘッドで実現した．近年，IoT の台頭により組込みソフトウェアの安全性がますます重要となっている．安全性を確保する方法の一つがパーティショニングであり，メモリ保護はその重要な要素であるが，既存のメモリ保護 OS ではメモリ保護により無視できないオーバヘッドが発生する場合がある．本研究は一般的なメモリ保護 OS で用いられる MPU を用いる代わりに TrustZone のハードウェア支援機能を利用し，ドメイン間遷移のレイテンシを低減する方法を提案する．提案手法を用いることにより，既存のメモリ保護 OS と同様のメモリ保護を，より低いオーバヘッドで実現できることを示した．

Lightweight RTOS Utilizing TrustZone for ARMv8-M

Kawada Tomoaki[1]　Honda Shinya[1]

1. はじめに

近年，IoT の台頭により，インターネットに接続する組込みシステムが増加し，組込みシステムの限られたリソース制約の中で安全性を確保することがますます重要化している．組込みシステムで安全性を確保する方法としては，パーティショニング機構を導入し，単一システム内のあるソフトウェアの故障が他の箇所に波及したり，情報が漏洩することを防ぐことが一般的である．パーティショニングの方法の一つが，メモリ空間を保護対象毎にドメインと呼ばれる単位で分割しドメイン間のメモリアクセスを制御するメモリ保護であり，TOPPERS/HRP2 カーネルはこれを実装した RTOS（リアルタイム OS）の一つである [1]．

こうしたメモリ保護 OS は，MPU と特権・非特権モードを利用しメモリ保護を実現するが，これには無視できないオーバヘッドが伴うことが多い．実際，一部のサービスコールの実行時間が 2 倍近く増加することが報告 [1] されており，またコードサイズも増加し，小型のシステムではハードウェアコストへ影響することが考えられることから，より軽量なアプローチが求められている．

ARMv8-M は，ARM 社が 2015 年に発表した組込み向けのアーキテクチャである．ARMv8-M に追加された主要な機能の一つが，TrustZone for ARMv8-M であり，この機能によりシステムをハードウェアレベルで Secure/Non-Secure の 2 つの領域に分離し，セキュアなデータやコード，及びハードウェアに対する非セキュアコードからのアクセスを制御できる．

TrustZone for ARMv8-M の特徴は，Secure/Non-Secure 間の実行モード遷移に必要な処理をハードウェア側で自動的に行うことができ，ソフトウェアのオーバヘッド（実行時間，コードサイズ）を低減できることである．TrustZone for ARMv8-M のモデルは従来のメモリ保護 OS で一般的に使用されてきた特権・非特権モードのものと類似している為，これを活用することでメモリ保護 OS をより低いオーバヘッドで実現できると考えられる．

本研究では，メモリ保護機能を持たない RTOS である TOPPERS/ASP3 をベースとして，TrustZone for ARMv8-M による低オーバヘッドなメモリ保護を実現した ASP3+TZ（TrustZone for ARMv8-M 対応の ASP3）を開発した．さらに，開発した ASP3+TZ に対し評価実験を行

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University

Vol.2017-ARC-225 No.33
Vol.2017-SLDM-179 No.33
Vol.2017-EMB-44 No.33
2017/3/10

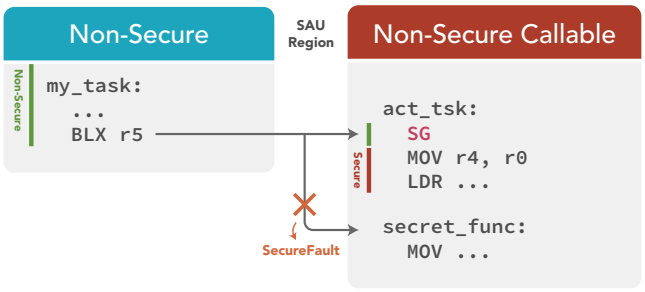


図 1　Non-Secure Callable 領域内にジャンプする際の動作
Fig. 1　Jumping into a Non-Secure Callable region

うことで，コード修正により，従来のメモリ保護 OS よりも低いオーバヘッドでメモリ保護 OS を実現できることを示した．最後に，ベースとした ASP3 からのコード変更量を評価することで，最小限のコード修正によりメモリ保護が実現できることを示した．

2. ARMv8-M アーキテクチャ

ARMv8-M アーキテクチャ [2] は，ARM 社が 2015 年に発表した，Cortex-M プロセッサファミリ向けのアーキテクチャである．ARMv8-M の主な特徴は，従来からある ARMv7-M の機能に加え，ARMv8-M Security Extensions（場合によっては Cortex-M Security Extensions と呼ばれる場合もある．以下 CMSE と略す）に対応したことである．CMSE は TrustZone for ARMv8-M に基づいた技術であり，システムをハードウェアレベルで Secure/Non-Secure に分離することで，Non-Secure がアクセス可能な領域を制限し，高水準のセキュリティを確保できる．

2.1 Cortex-M Security Extensions

CMSE では SAU (security attribution unit) 及び IDAU (implementation defined attribution unit) により，各メモリアドレスのセキュリティ属性を (a) Secure (Secure Mode のみアクセス可能), (b) Non-Secure Callable (Non-Secure からは制限付きでジャンプ可能; 後述), (c) Non-Secure (常にアクセス可能) のうち一つに設定できる．SAU による属性の指定方法は MPU と類似しており，各リージョンの上界・下界アドレス・属性をレジスタを経由して設定する．

SG (Secure Gateway) 命令は Non-Secure Callable 領域内で，Non-Secure からジャンプ可能なアドレスを示すために，関数の先頭で使用する命令である．SG にジャンプした場合，その時点で Secure Mode への遷移が自動で行われるが，それ以外の命令にジャンプした場合は即座に SecureFault 例外が発生する（**図 1**）．

TT (Test Target) 命令は指定されたメモリアドレスのセキュリティ・MPU 属性の判定を行う命令であり，Non-Secure から渡されたポインタの正当性の検証に使用できる．判定対象は単一のメモリアドレスであるが，結果に含まれる SAU, IDAU の対応するリージョンの番号を利用することで，アドレス範囲のアクセス権を判定できる．

3. CMSE を用いたメモリ保護 OS

IoT 向けの組込みシステムでは，ネットワーク上に存在する脅威からシステムを保護する為に，ネットワーク関連の機能を別のパーティションに分割することが望まれる．CMSE を用いてこれを実現する場合，以下のシステム構成方法が考えられる（**図 2**）．

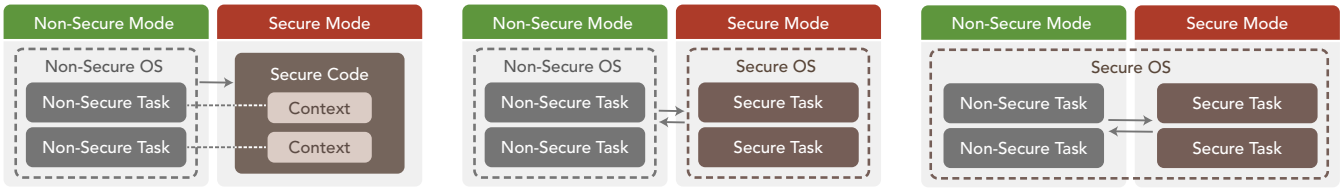
セキュアライブラリ方式　Secure 側はライブラリ形式として機能を提供し，その実行コンテキストの管理は Non-Secure 側で行う方式である．Secure 側のコンテキストは Non-Secure 側から明示的に要求して切り替える必要がある．この方式は ARM CMSIS（Cortex Microcontroller Software Interface Standard）で想定している方式である．CMSIS では Non-Secure 側から Secure 側のコンテキストを切り替えるための共通 API を定義し，Non-Secure 側の RTOS がこの API を使用することを推奨している [3]．

この方式は，プログラムに機密情報が含まれる場合に，想定される開発者以外がその情報にアクセスするリスクを軽減したい場合に適している．例えば，プログラムに暗号化鍵を埋め込む場合，暗号化機能のみを Secure 側で実装し，それ以外のコードは Non-Secure 側で実装し，Secure 側が提供する API を介して暗号化機能を利用する．Non-Secure 側の開発者はたとえデバッガを介しても Secure 側のコードにはアクセスできないため，暗号化鍵にアクセスできる開発者の数を最小限にでき，鍵が流出する危険を大きく減らせる．

デュアル OS 方式　Non-Secure 側と Secure 側のそれぞれで独立した OS を実行する方式である．この方式はセキュアライブラリ方式と類似しているが，Secure 側でも OS を動作させるため，それぞれの領域が実行コンテキストを独立して管理できる．

Secure 側でも OS を持つことにより，Secure 側で行える処理の幅は広がるが，両方の領域で OS を実行させるために，OS のコードが 2 重に必要となる．また，タスクの優先度の柔軟性が低く，混合スケジューリングを行うには OS の修正を要する．OS の境界を越えてサービスコールを発行することは簡単ではなく OS 間通信が必要であり，Secure 側と Non-Secure 側が密接に連携することが求められるシステムには向かない．

本研究では，以上で挙げた方法とは異なる，シングルシステムイメージ（SSI）方式を提案する．SSI 方式は，単一の OS・バイナリが Non-Secure と Secure の両側の制御を行う方式である．前に述べた方式と大きく異なる点は，Non-Secure と Secure でバイナリを分離せず単一のシステムイメージとして扱う点であり，セキュアライブラリ方式が想定するような，開発チーム内での機密情報の厳密な管



(a) セキュアライブラリ方式　(b) デュアル OS 方式　(c) シングルシステムイメージ方式

図 2　CMSE を用いたシステム構成方法
Fig. 2　System architectures for CMSE

理には向かない．

この方式は旧来のリングプロテクションを用いたメモリ保護 RTOS に類似している．他の方式と比べると単一のバイナリのみを開発するため開発効率が良く，さらに一つの OS が両方の領域を管理するため，柔軟なスケジューリングが可能である．セキュアライブラリ方式と比較すると，セキュアライブラリ方式では Non-Secure 側の RTOS がコンテキストスイッチする毎に Secure 側が提供する API を呼出して Secure 側のコンテキストを切り替える必要があるが，この方式では RTOS が両方のコンテキストを直接操作できる為，コンテキストスイッチによる遅延が少ない．また，デュアル OS 方式と比較しても，OS を 2 重に起動することによるオーバヘッドがない．

一方で，この方式では OS は両方の領域を考慮した処理を行う必要があるため，他の方式で OS を動作させる場合よりも，オーバヘッドが増加する可能性がある．例えば，Non-Secure 側でのサービスコールの呼出しは，Secure Mode の切り替えが必要となる為，実行時間への影響が考えられる．しかし，CMSE ではこうしたモードの切り替えを伴う呼出しをハードウェアの支援により高速化する機構を備えている為，同じことを旧来のメモリ保護 RTOS よりも遥かに少ないオーバヘッドで実現できると思われる．

4. 設計

ASP3+TZ の基本的な考え方は，カーネルとシステムドメインは常に Secure Mode で動作し，ユーザドメインのコードを Non-Secure Mode で動作できるようにすることで，カーネルドメインとユーザドメインの 2 種類の保護ドメインを実現することである．ユーザドメインのタスクは Non-Secure Mode で実行され，アクセス可能なメモリ領域は SAU 及び IDAU により定義されるセキュリティ属性によって制限される．サービスコールの呼出しにはシステムドメインの遷移が必要であるが，これは secure gateway を経由して行う（図 **3**）．

HRP2 では拡張サービスコールにより，システムドメインの関数へのアクセスをユーザドメインに提供していた．ASP3+TZ でも secure gateway を経由することで同様のことが可能であるが，それに加え，その逆となる，ユーザドメインの関数（非特権関数）をシステムドメインから CMSE を活用し高速に呼出すことも考えられる．この場合，ユーザドメインで発生した例外を安全にシステムドメインで処理する方法が必要になると思われる．

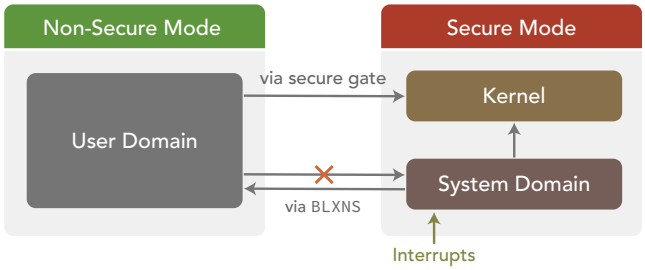


図 3　ASP3+TZ の概念図
Fig. 3　The concept of ASP3+TZ

ユーザドメインについては，複数個存在させ，ドメイン間のアクセスを制限したり，ドメイン毎に個別にペリフェラルやカーネルオブジェクトへのアクセス権を設定することも考えられる．パーティショニングを行う目的の一つはアプリケーションの故障による影響を局在化させることであり，機能ごとにドメインを細かく分割することで，故障発生時の影響を最小限にし，故障の分析に要する期間を短縮することができる．こうした考え方は基本的には旧来のメモリ保護 OS でも変わらないが，ASP3+TZ はそのオーバヘッドの低さの為，より積極的にドメインの分割を行えるようになる可能性がある．

また，HRP2 ではオーバヘッドの大きさから実装が避けられてきたユーザドメイン上の ISR の実装も検討している．AUTOSAR OS 標準規格 [4] では OS application 内で ISR を生成する方法について定義している一方で，特権・非特権モードを利用したメモリ保護 OS では結局システムドメインで起動した ISR からユーザドメインのエントリポイント関数を呼ぶ必要がある為，HRP2 では OS の機能としてはユーザドメイン上の ISR は提供していない．CMSE では Secure/Non-Secure で独立した割込みベクタテーブルを持つ為，ソフトウェアによる介在無しにユーザドメインで直接 ISR を起動することが可能である．

5. 実装

以上で述べた設計に基づいて，ASP3+TZ の実装を行っ

Vol.2017-ARC-225 No.33
Vol.2017-SLDM-179 No.33
Vol.2017-EMB-44 No.33
2017/3/10

た．ただし現時点で実装を行った範囲は部分的であり，複数ユーザドメインのサポートと，カーネルオブジェクトのアクセス制御（セマフォ等のオブジェクトにアクセスできるドメインを制限する機能），非特権関数，及びユーザドメイン上の ISR は実装を行っていない．これらの機能については，今後実装を検討している．

ASP3+TZ の実装方針は以下の通りである．

(1) メモリ保護機構を持たない RTOS の実装である ASP3 をベースとし，変更量を最小限にする．特に，カーネルのコアへの修正は最小限にし，可能な限りアーキテクチャ依存部のコアとは独立した部分のコード修正及び追加のみで実現する．
(2) 実行時オーバヘッドの最小化．

以上の方針を実現するために，CMSE の機能を積極的に利用した．具体的な実装項目は以下の通りである．

5.1 サービスコール用 Secure Gate の実装

多くのサービスコールはカーネル内部の状態を操作する為，Secure Mode で実行しなければならない．したがって，システムドメインから呼出す場合は従来通り通常の関数呼出しで十分である一方，ユーザドメインでこれらを呼出す為には Secure Mode への切り替えが必要である．CMSE では Non-Secure Callable リージョン内の SG 命令へジャンプすることにより，この切り替えを自動で行える．切り替えの際，LR（リターンアドレスレジスタ）の最下位ビットが 0 に変更され，リターン先が Non-Secure であることを表す値となる[*1]．リターンする際には，リターン先が Non-Secure である為，`BXNS` 命令を使用する．

Non-Secure Callable リージョンに配置した，先頭に SG 命令を持つ，サービスコールへの橋渡しを行う関数を secure gateway と呼び，対応するサービスコールの名前に `ns_`というプレフィックスを付加したものをその名前とした（ユーザドメインからはこの名前で呼出す）．**図 4** は secure gateway の例である．Secure gateway の SG 命令により Secure Mode への切り替えは自動的に行われるものの，ソフトウェアによりいくつかの追加の処理が必要である．

ポインタの検証　サービスコールにはポインタを渡すものが一部存在する．サービスコールの本体は Secure Mode で実行される為，引数として Secure リージョン内へのポインタを指定することにより許可されていない領域の値を不正に操作することが可能となってしまう．このため，CMSE が提供する `TT` 命令により，ポインタが指す領域のアクセス権情報を取得し，アクセスが禁止されている場合はエラーを返す処理が必要ある（図 4 右）．

ポインタの検証についてはサービスコールの本体側で実装することも考えられるが，呼び出し元ドメインにより分岐が必要な為，secure gateway での実装に留めた．

レジスタのクリア　Secure Mode と Non-Secure Mode でバンクされているレジスタは SP 等のごく一部に限られる．このため，サービスコールから復帰した後そのまま Non-Secure 側にリターンすると，レジスタに残った値から機密情報が漏れる恐れがある．図 4 の両方の例で `r0-r3` をクリアしているのはこれを防ぐ為である．

```
ns_act_tsk:
    sg
    push {lr}
    bl act_tsk
    pop {lr}
    mov r1, #0
    mov r2, #0
    mov r3, #0
    mov ip, #0
    msr apsr_nzcvq, r1
    bxns lr
```

```
ns_wai_flg:
    sg
    push {r4, lr}
    tst r3, #3
    tt r4, r3
    bne sg_perm_check2_fail
    tst r4, #TT_RESP_NSRW
    beq sg_perm_check2_fail
    bl wai_flg
    pop {r4, lr}
    mov r1, #0
    mov r2, #0
    mov r3, #0
    mov ip, #0
    msr apsr_nzcvq, r1
    bxns lr
```

左: ポインタを渡さないサービスコールの secure gateway
右: 第 4 引数の 32 ビット整数へのポインタのアクセス権を `TT` 命令により検証している．

図 4　Secure gateway の例
Fig. 4　An example of secure gateway

5.2 Non-Secure スタックの追加

各タスクに対し，Non-Secure Mode 用のスタック領域が追加で生成されるようにした．追加のスタック領域が必要である理由は複数ある．まず，ユーザドメインのタスクでは，タスクは Non-Secure Mode で起動されるが，サービスコールの実行中は Secure Mode に切り替わる．Non-Secure Mode で実行することを可能にするにはスタックを Non-Secure リージョンとする必要があり，同じ領域を Secure Mode からも使用してしまうと，サービスコールの実行中に他のユーザタスクからカーネルが使用しているスタック領域上のデータを操作される危険が生じる．もう一つの理由は，スタックポインタが Secure/Non-Secure Mode でバンクされているという ARMv8-M の仕様に起因するものである．このため，同じ領域を共有するには，サービスコールの先頭で Non-Secure 用スタックポインタを Secure 用スタックポインタにコピーする必要があり，その分わずかながら遅延が増加してしまう．

以上の理由から，カーネルコンフィギュレータに対して

[*1] この処理が自動で行われるのは，LR を Secure リージョンの任意の関数に変更して SG を呼出す攻撃が考えられる為である．LR を最下位ビットが 0 のアドレスしておけば，`BXNS` 命令でリターンする差は最初に Non-Secure Mode への切り替えが行われる為，Secure リージョンにジャンプしても例外が発生する．

Vol.2017-ARC-225 No.33
Vol.2017-SLDM-179 No.33
Vol.2017-EMB-44 No.33
2017/3/10

修正を行い，追加のスタック領域が生成されるようにした．

5.3 ディスパッチャの変更

直前のセクションで述べたように，ARMv8-M ではスタックポインタが Secure/Non-Secure Mode でバンクされており，コンテクストスイッチの際に両方のスタックポインタを保存・復元する必要があり，ディスパッチャの変更が必要となった．

ディスパッチの際，スタップポインタを復元するだけでなく，正しいプロセッサモード（Non-Secure/Secure）への切替を行う必要がある．タスクからディスパッチャに入るには PendSVC 例外を発生させるが，その際に LR レジスタに直前のプロセッサモードに依存した値（`EXC_RETURN`）が格納される．タスクに復帰させる際は，単純にその値を PC に書き込むだけでよく，プロセッサモードはその値に応じて自動的に適切な状態に設定される．

5.4 リンカスクリプト

SAU をコンフィギュレーションしメモリのセキュリティ属性を設定する為には，各ドメインに属する変数やコードが属するアドレス範囲の情報が必要である．このため，リンカスクリプトにより各ドメインのシンボルがまとまった範囲に定義されるようにし，その前後にリンカシンボルが生成されるようにした．カーネルのターゲット依存部の初期化コードでは，そのシンボルのアドレスを使用して，SAU のコンフィギュレーションを行う．

6. 評価

開発した ASP3+TZ のサービスコールの実行時間を計測し，メモリ保護機構を持たない RTOS の実装である ASP3 と比較することにより，オーバヘッドの評価を行った．その後，ASP3+TZ を実装する際に修正を要した行数を計測することで，コード変更量の評価を行った．

6.1 評価環境

評価用のターゲットボードとして，ARM 社製の Cortex-M Prototyping System（MPS2）を使用した．MPS2 は FPGA を搭載しており，FPGA イメージを書き込むことで各種 Cortex-M コアを使用した開発を行うことができる．本実験では FPGA イメージとして Cortex-M33 example IoT FPGA image for MPS2+のものを使用した．このイメージは ARMv8-M Mainline Profile の Cortex-M33 プロセッサと CoreLink SIE-200 System IP が含まれており，25MHz で動作する．プロセッサは AHB5 及び AHB5 Memory Protection Controller（MPC）を経由して基板上の計 8MB のシングルサイクル SRAM と 16 ビットバスで接続されており，ここにコードとデータを配置できる．また，UART インタフェースを内蔵しており，これを介して

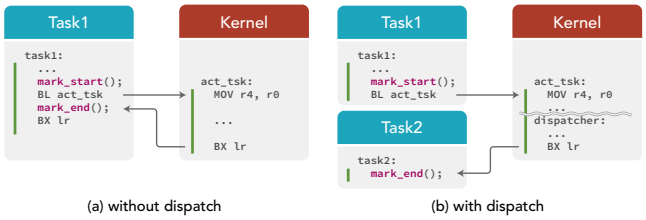


図 5 `act_tsk` の実行時間の計測方法
Fig. 5 The method for measuring the run time of `act_tsk`.

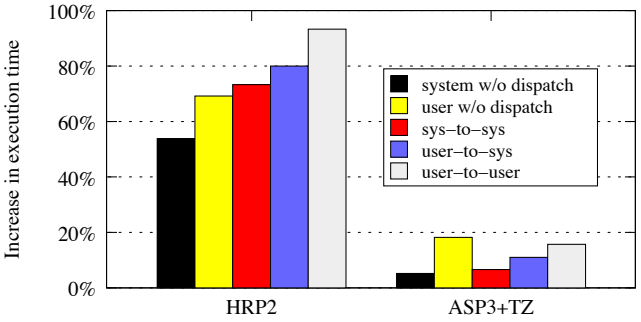


図 6 サービスコールの呼出しオーバヘッドの比較
Fig. 6 The overhead comparison of the service call.

PC と通信を行うことができる．

6.2 実行時オーバヘッド

ASP3+TZ におけるサービスコール `act_tsk` の実行時間を計測し，ASP3（非メモリ保護 OS）を基準としたときの増加率を求める評価を行った．`act_tsk` は TOPPERS 第 3 世代カーネル統合仕様書 [5] で定義されている，タスクを起動するサービスコールである．`act_tsk` は実行中タスクと対象タスクの優先度の相対関係によって，(a) ディスパッチが発生しない場合と (b) 発生する場合の 2 通りの動作を行う．このため，(a) の場合は `act_tsk` を呼出してからリターンするまでの時間，(b) の場合は `act_tsk` を呼出してから対象タスクの最初のステートメントが実行されるまでの時間を計測した．さらに，ASP3+TZ では呼出し元のドメイン（ユーザ/システム）の違いがあり，さらに (b) の場合については呼出し先のドメインの違いもあるため，各組合せについて計測を行った．

比較対象として，TOPPERS/ASP（非メモリ保護 OS）と HRP2 で同様の計測を行ったときの結果 [1] を元にして，HRP2 での実行時間の増加率を求めたものを使用した．呼出し元がシステム，呼出し先がユーザドメインの場合の計測結果がなく比較できない為，この組合せは計測を行っていない．なお評価環境が本研究で用いたものと異なるが，これが実行時間の増加量でなく増加率を用いた理由である．

6.2.1 結果・考察

各メモリ保護 RTOS（ASP3+TZ/HRP2）の `act_tsk` の実行時間の増加率を図 **6** に示す．HRP2 の結果は [1] に基づく．

Vol.2017-ARC-225 No.33
Vol.2017-SLDM-179 No.33
Vol.2017-EMB-44 No.33
2017/3/10

```
  200044:   f000 b82c   b.w 2000a0 <__ns_act_tsk_veneer>
002000a0 <__ns_act_tsk_veneer>:
  2000a0:   f85f f000   ldr.w   pc, [pc]
10000114 <ns_act_tsk>:
10000114:   e97f e97f   sg
10000118:   b500        push    {lr}
1000011a:   f002 fb89   bl  10002830 <act_tsk>
10002830 <act_tsk>:
```

`__ns_act_tsk_veneer` は long jump (`b` 命令で参照不可能な範囲へのジャンプ) を実現する為にリンカが自動生成した関数である．

図 7 ASP3+TZ のユーザドメインから `act_tsk` を呼出した際に実行される命令列

Fig. 7 The instruction sequence executed during the call to `act_tsk` from the user domain in ASP3+TZ.

図 6 の結果は，ASP3+TZ はいくつか制約があるものの，HRP2 と比較して大幅に低いオーバヘッドでメモリ保護を実現できることを示している．

このような差が生じた理由の一つとして，ユーザドメインからサービスコールを呼出す際に実行される命令数に大きな差があることが考えられる．どちらのドメインから呼出した場合でも最終的にはカーネル内にあるサービスコールの本体に辿り着くが，ユーザドメインから呼出す場合はモードの遷移等に追加の処理が必要である．HRP2 ではこの処理に必要な命令数は 33 命令であるが，ASP3+TZ ではそれより大幅に少ない 5 命令で済んでいた（**図 7**）．4 説でユーザドメインの細分化により故障を局在化できることについて述べたが，この結果は，大きなオーバヘッドを生じることなくより多くのコードをユーザドメインで実行することが可能であり，その結果，より厳密なアクセス権限の設定が可能になることを意味している．

6.3 カーネルのコード変更量

ASP3+TZ の実装方針は，最小限の変更で，メモリ保護 OS を実現することであると述べた．そこで，この方針の達成度合いを検証する為に，開発した ASP3+TZ と元となった ASP3 のソースコードを行単位で比較することで，コード変更量の評価を行った．

結果を表 1 に示す．アーキテクチャ依存部と非依存部の両方についても，変更量は 1 割程度に留まっていることが確認できる．特に，非依存部の変更量は非常に少ない．この非依存部の変更はカーネルコンフィギュレータへの修正で，Non-Secure スタックを生成するようにするためのものである．以上の結果から，カーネルのコード変更量を最小限にするという目標は十分に達成できていると考えられる．

表 1 カーネルのコード変更行数

Table 1 Kernel code additions and deletions

コード	ASP3 SLoC	追加	削除
アーキテクチャ依存部	1884	233	23
非依存部	7502	25	3

7. おわりに

IoT 向けの組込みシステムは高い安全性が求められており，パーティショニングによりドメイン間のアクセスを制限し，故障の波及や情報の漏洩を防ぐことが重要である．しかし，従来型のメモリ保護 RTOS はメモリ保護の導入により大きなオーバヘッドを生じるという欠点があった．そこで本論文では，従来のメモリ保護 RTOS の欠点を克服する，TrustZone for ARMv8-M を用いた軽量なメモリ保護 RTOS ASP3+TZ の実現方法について述べた．ASP3+TZ はメモリ保護機構を持たない RTOS である ASP3 をベースとしている．開発した ASP3+TZ に対しサービスコールのオーバヘッドの評価を行い，既存のメモリ保護 OS と比較して非常に少ないオーバヘッドでメモリ保護を実現できることを示した．さらに，ベースとした ASP3 からのコード修正量を比較を行うことで，最小限の変更で ASP3+TZ を実現できることを示した．

現時点で残されている課題は，HRP2 にあるような時間保護やオブジェクトのアクセス権制御に対応することである．これらはパーティショニングに必要不可欠な要素であり，若干のオーバヘッドを伴っても実装が望まれる．また，TrustZone for ARMv8-M では割込みに対しても Secure/Non-Secure 設定が行え，ハードウェアにより自動的にモードの遷移が行えることから，これを利用して割込みハンドラをユーザドメインで直接実行する方法についても検討する予定である．

謝辞 本研究の一部は JSPS 科研費 JP26330062 の助成を受けたものです．

参考文献

[1] 石川拓也，本田晋也，高田広章：静的なメモリ配置を行うメモリ保護機能を持ったリアルタイム OS，コンピュータソフトウェア，Vol. 29, No. 4, pp. 161–181 (2012).
[2] Ltd., A.: ARMv8-M Architecture. `https://www.arm.com/products/processors/instruction-set-architectures/armv8-m-architecture.php`.
[3] Ltd., A.: Application Note 291: Word final word Using TrustZone on ARMv8-M. `http://www.keil.com/appnotes/docs/apnt_291.asp`.
[4] Automotive Open System Architecture GbR: Specification of Operating System. `http://www.autosar.org/standards/classic-platform/release-43/software-architecture/system-services/`.
[5] NPO 法人 TOPPERS プロジェクト：TOPPERS 第 3 世代カーネル（ITRON 系）統合仕様書，release 3.0.0 edition (2016). `https://www.toppers.jp/docs/tech/tgki_spec-300.pdf`.