Vol.2009-EMB-14 No.8
2009/7/24

ARMv6アーキテクチャを用いたメモリ保護RTOSのユーザスタック保護の設計と評価

中嶋 健一郎[†1] 山田 真大[†1] 長尾 卓哉[†1]
山崎 二三雄[†1] 武井 千春[†1]
本田 晋也[†1] 高田 広章[†1]

組込みシステムでは，これまでメモリ保護機能を実現する際のオーバヘッドやMMUを利用することによってリアルタイム性が低下するという問題があったため，メモリ保護機能を持つことが少なかった．しかし近年では組込みソフトウェアが大規模複雑化してきたことによって，メモリ保護の必要性が増大しつつある．

本研究では，ARMv6 アーキテクチャのプロセッサ上で，メモリ保護機能を持つRTOS である TOPPERS/HRP2 カーネルにおいてユーザスタックを保護する方法を検討し，ページテーブル書換え方式，red zone 方式，ARM ドメイン方式の 3 つの手法を提案し，比較した．その結果，ページテーブル書換え方式はリアルタイム性に難があること，red zone 方式，ARM ドメイン方式は似た性能を示すが信頼性，ハードウェアの制約に差があることがわかった．

Design and Evaluation of Memory Protection for Realtime Operating System

Kenichiro Nakajima ,[†1] Masahiro Yamada ,[†1]
Takuya Nagao ,[†1] Fumio Yamazaki ,[†1]
Chiharu Takei ,[†1] Shinya Honda [†1]
and Hiroaki Takada[†1]

There is rarely memory protection in most embedded systems. Because the most processor did not have the hardware to realize memory protection function and the software has been small in size enough to verify completely. But recently these assumptions are not right. Many processors has memory protection function. And the embedded system size and complexity is increasing.

We discussed methods to realize a task stack protection of TOPPERS/HRP2 kernel, the RTOS has memory protection function, on the ARMv6 architecture. We experienced by the updating pagetable, the using red zone, and the using ARM domain. As a result, we identify that the updating pagetable cannot have enough realtime ability, and that the using red zone and the using ARM domain have similar realtime ability, but differ in reliability and hardware restriction.

1. はじめに

組込みシステムでは，これまでメモリ保護機能を持つ RTOS を採用することが稀であった．なぜなら，利用するプロセッサがメモリ保護機能を実現するために必要なハードウェア（Memory Management Unit（MMU）など）を持たないことが多かったことや，ソフトウェア自体が小規模であり高い信頼性を確保するための全体の検証が比較的容易であったことのためである．しかし近年ではメモリ保護機能を持つプロセッサが増加してきたことや，組込みソフトウェアが大規模複雑化してきたためにメモリ保護機能の必要性が高まりつつある．

OS がメモリ保護機能を持つことで，OS 上で動作するアプリケーションが誤動作した場合に，その誤動作が OS 自身や他のアプリケーションに障害を引き起こすのを防ぐことができる．また，メモリ保護機能を持つことによって，ソフトウェアに潜在しているバグを早期に発見でき，テスト期間の短縮・テストコストの削減を期待できる．加えて，信頼性を確保するために行う検証もアプリケーション毎に考えればよいため容易となる．

汎用 OS もメモリ保護機能を持つが，OS として性格の違いからメモリ保護の特徴もまた異なる．例として，多くの汎用 OS ではアドレス変換を使ったプロセスモデルを採用しているため，アドレス変換機能は必須であることが挙げられる．これに対し，メモリ保護機能を持つ RTOS である，TOPPERS/IIMP カーネルでは Memory Protection Unit（MPU）を利用することを想定しており，アドレス変換機能は利用していない．一般に，リアルタイム OS においては，MMU は実行時間の予測可能性が低く，リアルタイム性の確保が難しいといわれている[1)]．しかし，近年使われている高スペックなアプリケーション向けのプロセッサでは，MPU ではなく MMU を持つことが多くなってきている．

そこで，本研究では，TOPPERS/HRP2 カーネルの仕様に従って，ユーザスタックの保護を行う方法を検討した．具体的には，ARMv6 アーキテクチャ[2)] のメモリ保護機構を利用

†1 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
Center for Embedded Computing System, Nagoya Univ.

して，ページテーブル書換え方式，red zone 方式，ARM ドメイン方式の 3 つの手法を実装した．この実装に対して，各方式が持つオーバヘッドを評価するため，メモリ保護機能を持たない RTOS である TOPPERS/ASP カーネルとこれらの各方式において，タスク切換え時間，システムコール呼び出し時間を計測し，結果を比較した．その結果，ページテーブル書換え方式はリアルタイム性に難があること，red zone 方式，ARM ドメイン方式は似た性能を示すが信頼性，ハードウェアの制約に差があることがわかった．

本研究では組込みシステムで広く使われている ARM プロセッサを利用しているため，汎用性は高い．Linux などの汎用 OS ではパフォーマンスのためにプロセス内のスレッド間でスタックの保護を行っていないが，本研究の成果を適用することでパフォーマンスを悪化させず信頼性を高めることが期待できる．

本論文では，まず 2 章で TOPPERS/HRP2 カーネルの特徴について述べる．3 章で関連研究について述べる．4 章で本研究で使用した ARMv6 アーキテクチャの仮想メモリシステムについて述べる．5 章で TOPPERS/HRP2 カーネルでの ARMv6 アーキテクチャのメモリ保護機構の使用方法について述べる．6 章で本研究で検討したユーザスタック保護方式について述べる．7 章で検討した手法の評価を行う．8 章で本論文のまとめを行う．

2. TOPPERS/HRP2 カーネルについて

この章では本研究で実装を行ったリアルタイム OS の仕様について説明する．

TOPPERS/HRP2 カーネルとは，μITRON4.0/px 仕様をベースとして TOPPERS プロジェクトで拡張および実装を行った，メモリ保護機能を持つ高信頼システム向けリアルタイム OS である．TOPPERS/HRP2 カーネルの実装は，TOPPERS/ASP カーネルがベースとなっている．

TOPPERS/HRP2 カーネルは，メモリ保護機能，オブジェクトアクセス保護機能，時間保護機能（オーバランハンドラ機能）を持つ．メモリ保護機能は，ユーザタスクにおいてアクセスしてはならないメモリ領域へのアクセスの防止を行う機能である．オブジェクトアクセス保護機能は，システムコール発行時に発行元の権限をチェックし，アクセスしてはならないカーネルオブジェクトへのシステムコールによるアクセスを防止する機能である．時間保護機能は，タスクが指定されたプロセッサ時間を消費したことを検出し，例外処理のためのオーバランハンドラを起動する機能である．

TOPPERS/HRP2 カーネルの持つメモリ保護機能は汎用 OS のメモリ保護機能と比較すると以下の 3 点で異なっている．

1 点目として，リアルタイム性の確保と低オーバヘッド・低リソース使用を目標としていることが挙げられる．汎用 OS ではリアルタイム性を犠牲にしつつスループットを向上させることがある．また，OS のオーバヘッドが大きくなってしまった場合には，ハードウェアの性能を向上させることで対処することを考えることがある．TOPPERS/HRP2 カーネルでは，リアルタイム性を阻害しないことや限定されたハードウェアで実行できることを要件としている．

2 点目として，アドレス変換を行う必要がないことが挙げられる．多くの汎用 OS ではソフトウェアの開発を容易にし，メモリ利用効率を上げるために，アドレス変換を使ったプロセスモデルを採用している．しかし，組込みシステムの開発においては開発ツールが対応していないことが多く，アドレス変換を行うことが必ずしも有利ではないため，TOPPERS/HRP2 カーネルではアドレス変換を行わない．そのため，アドレス変換機能を保有する MMU ではなく，MMU よりシンプルな MPU によっても実現でき汎用性が高くなっている．

3 点目として，メモリ配置が静的であることが挙げられる．TOPPERS/HRP2 カーネルでは事前にメモリ配置が全て判明しているため，キャッシュの利用に適したプログラムの配置を行うことや，保護範囲を事前に計算しておくことでオーバヘッドを低減させることなどの最適化が可能となる．

2.1 メモリ保護・オブジェクトアクセス保護の基本的な考え方

オブジェクトアクセス保護は，アクセス対象がアクセス実行元となるアクセス主体を限定することで保護を実現する．

アクセス主体とは，アクセスを実行するタスクや割込みハンドラなどの処理単位を指す．アクセス対象とは，アクセスされるタスクやセマフォなどの各種カーネルオブジェクトを指す．アクセス対象毎に，どのようなアクセス主体が通常操作，管理操作，参照操作といった操作毎にアクセスが可能であるかを示すビットパターンが OS に用意される． **図 1** にオブジェクトアクセス保護の例を示す．アクセス対象であるタスク 1 は，アクセス主体であるタスク 1 からのアクセスは受け入れるが，タスク 2 からのアクセスは禁止することを示している．また，アクセス対象であるメモリは，アクセス主体であるタスク 2 からのアクセスは禁止しているが，ISR1 からのアクセスは受け入れることを示している．

2.2 保護ドメイン

カーネルオブジェクトをまとめたものを保護ドメインと呼ぶ．

アクセス許可パターンは保護ドメインを単位として，いずれの保護ドメインからどのようなアクセスを行うことができるかを設定する．

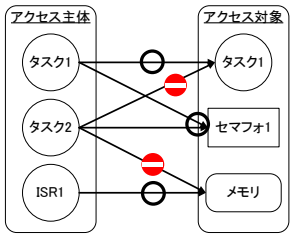

図 1 オブジェクトアクセス保護の例
Fig. 1 an example of access control

TOPPERS/HRP2 カーネルの持つ保護ドメインには以下の種類がある．

- カーネルドメイン
  システム内に一つだけ存在する．
  すべてのカーネルオブジェクトに対して，すべての操作/アクセスが許可される保護ドメインである．特権モードで実行される．
- ユーザドメイン
  システム内に複数存在してもよい．
  カーネルオブジェクトに対する操作/アクセスに制限のある保護ドメインである． 非特権モードで実行される．
- 共有ドメイン
  システム内に一つだけ存在する．
  ユーザドメインからの操作/アクセスを制限しないドメインである．非特権モードで実行される．

保護ドメインのイメージを **図 2** に示す．

2.3 メモリ保護の設定

メモリ保護は，カーネルドメインの設定とユーザドメインの設定で異っている．カーネルドメインは全てのドメインに対してアクセスを行うことができる．ユーザタスクから見た場合の，自ドメイン内の他タスク，他ドメインの他タスクに対するアクセス設定を **表 1** に示す．

TOPPERS/HRP2 カーネルの仕様から，同一のユーザドメイン内に存在する関数を呼び出すことができなければならない．そのため，同一のユーザドメイン内では text 領域及び rodata 領域，data 領域，bss 領域はいずれのタスクに対しても可視でなければならない．

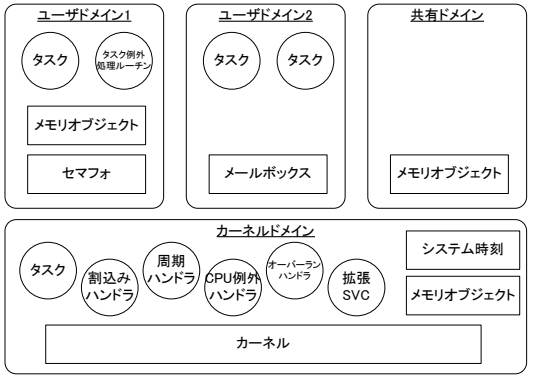

図 2 保護ドメインのイメージ
Fig. 2 an image of protection domain

タスクの持つスタックにはシステムスタックとユーザスタックの 2 種類がある．カーネルタスクはシステムスタックのみを持つ．ユーザタスクはシステムスタックとユーザスタックを持つ．

表 1 では，自ドメイン内の自タスクのユーザスタックにはアクセス可能，他タスクの持つユーザスタックへのアクセスが?となっている．これは，現時点では TOPPERS/HRP2 カーネルの仕様として他タスクのユーザスタックに対するメモリ保護が検討中の事項であることを示している．ユーザスタックの保護は必要と考えられるが，完全な保護を行うことによるオーバヘッドの増加が懸念されるため，本研究の結果を参考に検討していきたいと考えている．他ドメインのタスクに対しては，全てのアクセスが禁止である．

3. 関 連 研 究

この章では，他の OS におけるメモリ保護方式として，Linux[3]，Rtems[4]，AUTOSAR OS[5] について検討した結果を述べる．

3.1 Linux

Linux では，プロセス毎にプロセス ID と対応してページテーブルを一つ持ち独立した仮想メモリ空間を確保している．そのため，プロセス間ではお互いのメモリに干渉することなくプログラムを実行することができる．

しかし，プロセス内で動作するスレッドは高速なコンテキスト切り替えを実現するため

表 1 ユーザタスク間のアクセス可能性の仕様
Table 1 access permission during user tasks

	自ドメイン		他ドメイン
領域	自タスク	他タスク	他タスク
text	○	○	×
rodata	○	○	×
data	○	○	×
bss	○	○	×
ユーザスタック	○	?	×

○: アクセス可, ×: アクセス不可

に，親プロセスと同一のページテーブル上で動作するようになっている．このため，スレッド間ではスタックは保護されていない．

動的メモリ管理を行うためアドレス変換は必須であり，物理アドレスが連続しているかどうかは基本的に保障されない．スタックの底には，red zone と呼ばれる，スタックオーバフロー検知のためのアドレス範囲が設定してあり，この部分にアクセスを行った場合にスタックオーバフローとして扱う．

3.2 Rtems

Rtems は組込み向けリアルタイム OS である．Rtems では，特権モードでのみアクセス可能な領域と非特権モードであってもアクセス可能な領域を設定している．こうすることで，ユーザアプリケーションが不用意に重要な領域を破壊してシステムを停止させないようにしている．一方，タスク間では保護が存在しないため，アクセスを行うことができるようになっている．

3.3 AUTOSAR OS

AUTOSAR OS は自動車制御用リアルタイム OS である．AUTOSAR OS は仕様であり，どのように実装するかは規定されていない．

AUTOSAR OS では，特権モードでのみアクセス可能な領域と非特権モードであってもアクセス可能な領域を設定している．アプリケーションは信頼アプリケーションと非信頼アプリケーションに分類される．

信頼アプリケーションは特権モードで動作する．信頼アプリケーションは全てのメモリ，オブジェクトにアクセスが可能であり，他のアプリケーションから呼び出すことができる信頼関数を公開する．

非信頼アプリケーションは非特権モードで動作する．割込みサービスルーチンも非信頼アプリケーションとなる．非信頼アプリケーションはあらかじめ許可されたメモリ，オブジェクトにのみアクセスすることが可能である．他の非信頼アプリケーションのコード，データ，スタックを読み書きすることはできない．同一のアプリケーション内では，共有のデータを読み書きすることは可能となっている．他のタスクのデータ，スタックへのアクセスは禁止してもよい．

AUTOSAR OS にはスタックオーバフローを検出する機能があり，コンテキストスイッチの時点でタスク/ISR のスタックがオーバフローしていないかをチェックする．もしスタックオーバフローが発生していたならば，ProtectionHook を呼び出して対応することとなっている．

4. ARMv6 アーキテクチャ（The ARM Architecture Version 6）の仮想メモリシステムについて

本章では，ARMv6 アーキテクチャの仮想メモリシステムについて本論文で関連する事項について説明する．

4.1 Address Space IDentifier（ASID）

ARMv6 アーキテクチャでは，テーブルエントリにおいて ng ビットを立てることで，ページ単位でメモリ空間を非共有であると示すことができる．この非共有のメモリ空間を多重化するために，ASID という名前の識別子を利用することができる．ASID は非共有であると設定された空間においてのみ有効である．

Translation Lookaside Buffer（TLB）にはページテーブルエントリの持つ，メモリ空間毎のアクセス許可やキャッシュ属性などがキャッシュされる．

ASID を利用しない状態で，タスク切り替え時にページテーブルエントリを変更すると，変更前の情報を TLB 上に残すことはできないため，変更したページテーブルエントリの保持するメモリ空間に対応する TLB エントリは全てフラッシュしなければならない．しかし，フラッシュした TLB が対応しているアドレスに対し，後にアクセスした際には TLB ミスを引き起こすことになる．TLB ミスが発生した際には実メモリに 1-2 回程度アクセスする必要がありコストが高いため，システムのパフォーマンスを向上させるためにはできる限り TLB フラッシュを回避しなければならない．

これに対し，ASID を利用した状態では，仮想アドレスを 32 ビットから ASID の持つ 7 ビット分拡張したものとして扱うことができるようになる．そのため，タスク切り替え時にページテーブルエントリを変更したとしても，ASID が異なるならば別の空間として TLB

にキャッシュされる．よって，明示的な TLB フラッシュは不要となり，TLB ミスの回数を低減させることができるようになる．

4.2 ページテーブル

ARMv6 アーキテクチャの MMU では 2 段階のハードウェアテーブルウォークを行うことができる．2 段階それぞれのテーブルウォークのために，変換テーブルを持つ必要がある．

扱うことのできるページサイズは，16MB，1MB，64KB，4KB である．これらのページサイズは仮想アドレス内で混在させることが可能である．16MB もしくは 1MB のページサイズを持つ場合，この空間をセクションと呼ぶ．64KB，4KB のページサイズを持つ場合，この空間をページと呼ぶ．

1 段階目のテーブルウォークのためのテーブルをセクションテーブルと呼ぶ．このテーブルは 1MB または 16MB のセクション，もしくは 2 段階目のテーブルの先頭アドレスを保持することができる．

2 段階目のテーブルウォークのためのテーブルをページテーブルと呼ぶ．このテーブルでは，4KB もしくは 64KB のページを保持することができる．

1MB セクションとして利用する際のセクションテーブルの設定を**図 3** に，4KB ページとして利用する際のセクションテーブルとページテーブルの設定を**図 4** に示す．

ページテーブルを利用する際には，セクションテーブルでしか設定できない項目があるため，1MB 単位でのみ設定できる属性と個々のページテーブルエントリで 4KB 毎に設定できる属性があることに注意をする必要がある．

4.3 ARM ドメイン

ARM アーキテクチャでは，ドメインと呼ぶメモリの保護機能が存在するが，保護ドメインとの混乱を避けるため本論文ではこの機能を ARM ドメインと呼ぶことにする．

セクションエントリに ARM ドメイン番号を登録することで，仮想メモリ空間を 16 個の領域まで分割することができ，それぞれの領域に対して一度に，無条件でアクセス許可する領域，無条件でアクセス禁止する領域，ページテーブルの設定に従う領域と 3 種類に設定することができる機能である．

5. TOPPERS/HRP2 カーネルでの ARMv6 アーキテクチャのメモリ保護機構の使用

本章では，TOPPERS/HRP2 カーネルでの ARMv6 アーキテクチャのメモリ保護機構の使用方法について説明する．ここではユーザスタック以外の部分について説明する．

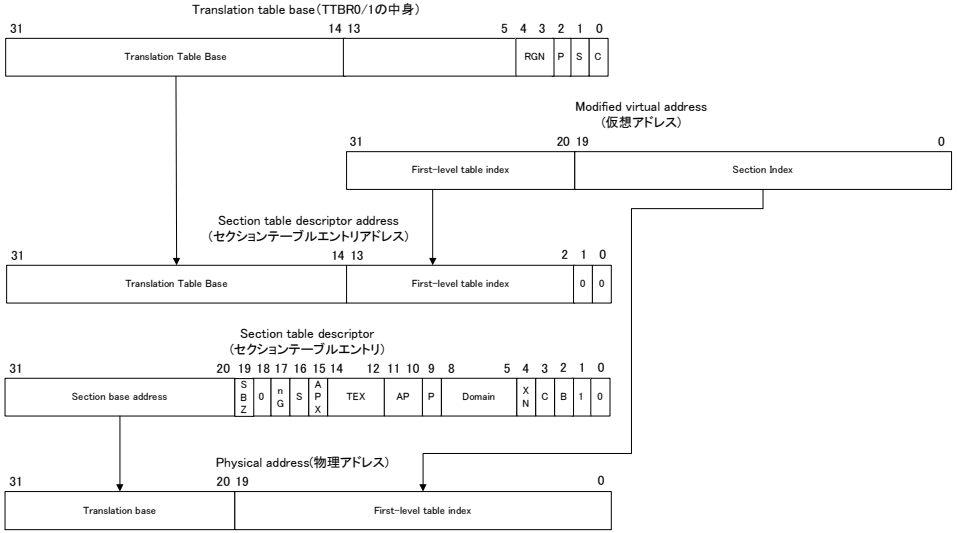

図 3 1 段階テーブルウォーク
Fig. 3 level 1 table walking

5.1 ASID の使い方

TOPPERS/HRP2 カーネルでは，ASID をドメイン ID（ドメインを区別する ID）として利用する．

ドメイン ID はユーザドメイン毎に割振っているため，ユーザドメインを切り替える必要が発生した場合にはドメイン ID も同時に変更する．これと同時に ASID を変更することで，それまでアクセスしていた空間に対してユーザドメイン切り替え後アクセスした場合には，異なる ASID が設定されているため TLB 内にキャッシュされている以前のエントリにはヒットしないようにすることができる．その結果，TLB をフラッシュする必要がないため，高速にユーザドメインを切り替えることができる．

5.2 ページテーブルの持ち方

本研究の実装ではユーザドメイン毎にセクションテーブルを保持している．各ドメインの各領域における属性を**表 2** に示す．

TLB エントリはテーブルエントリ毎に消費されるため，カーネルドメインについては 1MB セクションを使ってメモリ保護を行うこととした．カーネルドメインは独自の変換テー

Vol.2009-EMB-14 No.8
2009/7/24

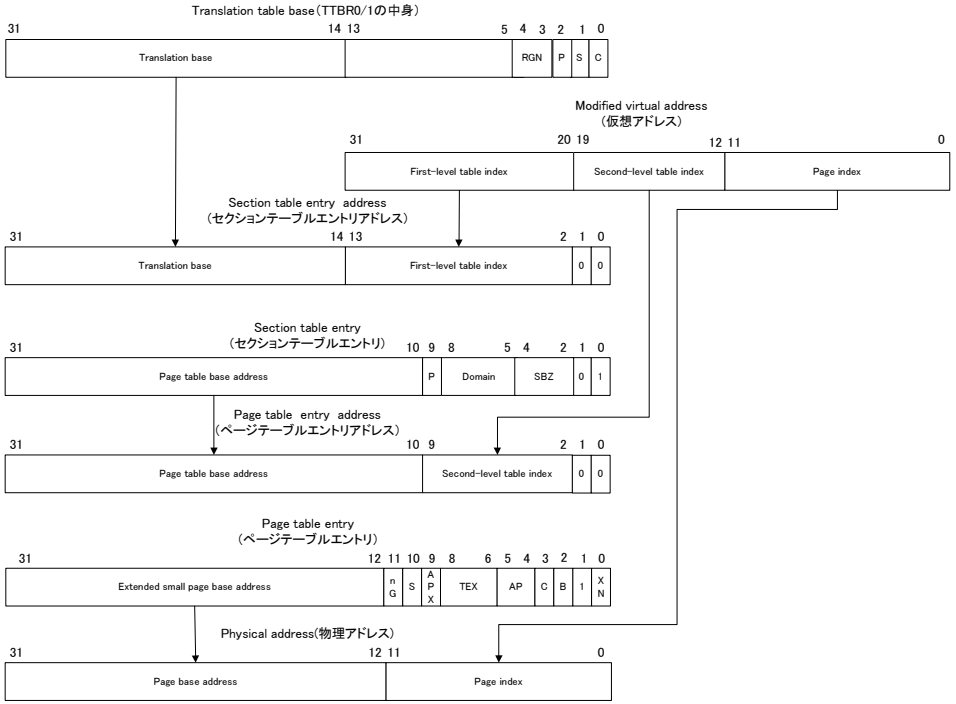

図 4 2 段階テーブルウォーク
Fig. 4 level 2 table walking

ブルを持たず，各ユーザドメインの変換テーブルにおいて，特権アクセスとして振舞う．これを実現するために，各ユーザドメインの持つ変換テーブル上で同一の制限を持つように設定を行った．

カーネルドメインの text 領域は実行可能，rodata 領域は実行禁止と個別に設定した方が信頼性という点では優れているように思えるが，それぞれにセクションを 1MB ずつ割り当てた場合メモリの使用効率が悪化する．そのため，rodata 領域は text 領域に連続して同一セクション内に配置する．同一セクション内では設定も同一になるため，rodata 領域には実行禁止属性を付けることはできない．そのため，誤って rodata 領域を実行してしまった場合に例外が発生せずカーネルが暴走する可能性がある．しかし，カーネルドメインの rodata 領域はカーネルのみがアクセス可能であるため，もし rodata 領域を実行してしまい

表 2 テーブルエントリの属性
Table 2 attribute of table entry

	kernel	user	shared
text	ro/na	ro/ro, ng	ro/ro
rodata	ro/na	ro/ro, ng, xn	ro/ro, xn
data	ro/na,xn	rw/rw, ng, xn	rw/rw, xn
bss	ro/na,xn	rw/rw, ng, xn	rw/rw, xn
user stack		rw/rw, ng, xn	

ro: 書き込み禁止, rw: 読み書き可, na: アクセス禁止,
ng: 非共有, xn: 実行禁止
特権アクセス/非特権アクセスとして示す.

エラーを検出したとしても回復することはできず，メモリの使用効率を悪化させて実行禁止属性を付ける積極的な理由がない．

一方ユーザドメインおよび共有ドメインでは，メモリ利用効率を上げるため 4KB ページを使って各領域を作成し，メモリ保護を行うこととした．

ユーザドメインの保持する全ての領域については，ng ビットを立て非共有とした．更に，各ユーザドメインの領域は，自らが所属するユーザドメインの変換テーブルでのみ有効であり，それ以外のユーザドメインの変換テーブルでは特権アクセスのみ可能として設定した．

共有ドメインの領域については，全てのドメインの変換テーブルにおいて同一の設定を行うこととした．この領域は ng ビットを設定せず ASID の影響を受けない．

デバイスやシステムレジスタなどのメモリマップド領域については，キャッシュ不可，バッファ不可として設定した．これらの領域はカーネルドメインに配置し，ユーザドメインからのアクセスは許可しないこととした．

5.3 ドメイン切り替え処理

ドメインの切り替え処理は，タスク切り換えを行うディスパッチャで実行する．

まず，切り替え先のタスクが所属するドメインを調べる．切り替え先がカーネルドメインの場合には，以下の処理を一切行わず通常のタスク切換え処理を実行する．切り替え先がユーザドメインの場合には，切り替え先ユーザドメインの持つ ASID の調査を行う．

ASID に変化がなければスタック切り替え処理を行う．ASID に変化があるならば，セクションテーブルを切り替えた後 ASID の登録を行う．

5.4 システムコール呼び出し

TOPPERS/HRP2 カーネルでは，システムコールの実行を特権モードで行う．そのため，システムコールの呼び出しは，非特権モードから特権モードへの切り替えを行う svc 命令

を実行することで行う．カーネルドメインは特権モードで動作しているが，svc 命令を実行しシステムコールを呼び出すこともできる．ユーザドメインは非特権モードで動作しているため，必ず svc 命令を利用してシステムコールを呼び出さなければならない．

スタックポインタレジスタは，ユーザモードと特権モード間でバンクレジスタとなっており，特権モードのスタックポインタレジスタはタスク固有のシステムスタック領域を指している．よって，特権モードに移行した際には自動的にシステムスタック領域を使用するようになっている．このとき，ユーザドメインの切り替えは行わないため，ドメイン切り替え処理は不要である．

6. ユーザスタック保護方式

ユーザスタックを保護するための方式として，本研究では以下の以下の 3 種類を検討した．

- ページテーブル書換え方式
- red zone 方式
- ARM ドメイン方式

本章ではこれらについてそれぞれ説明を行う．

6.1 ページテーブル書換え方式

6.1.1 概　　要

この方式は，タスク切り替えの都度切り替え元のユーザタスクのユーザスタックを無効にし，切り替え先のユーザタスクのユーザスタックを有効になるようページテーブルを書換えることで，ユーザスタックの完全な保護を実現する方式である． **図 5** にタスク切換え時のアクセス許可の変化を示す．タスク 1 を実行中には，タスク 2 のユーザスタックにはアクセスを行うことができず，タスク 2 を実行中には，タスク 1 のユーザスタックにはアクセスを行うことができないようになっている．

この方式を用いるメリットは，ユーザスタックの確実な保護が実現できることである．しかしその反面，タスク切り替えのオーバヘッドが大きくなってしまうというデメリットがある．これは，ページテーブルを書換える必要があることと，一般的にユーザスタック領域のページエントリの書換え後，TLB の内容を新しい内容で更新するために，書換えたアドレス範囲のページテーブルエントリに対し TLB フラッシュを行わなければならないためである．

6.1.2 実　　装

各ユーザドメインにセクションテーブルを割り当て，ページテーブルを使ってユーザスタックのアクセス権限を管理する．タスクが切り替わる際には切り替え元タスクのユーザスタック領域に対応するページテーブルエントリを非特権アクセス禁止状態にし，切り替え先タスクのユーザスタック領域に対応するページテーブルエントリをフルアクセス可能と変更することで実現する．

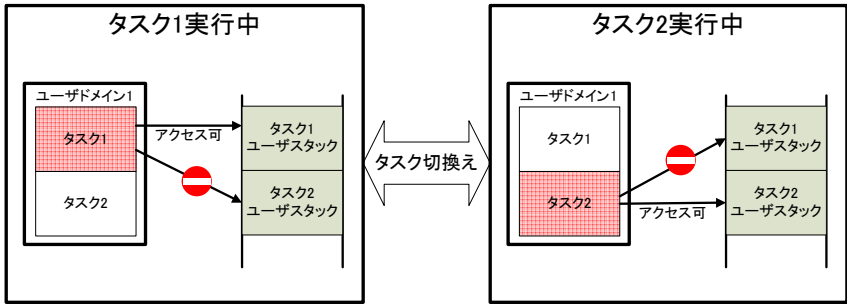

図 5 ページテーブル書換え方式のユーザスタック保護
Fig. 5 user stack protection using the updating pagetable

タスクを切り替える際には，ユーザスタックを切り替える必要があり，ページテーブルの書換えとそれに伴う TLB フラッシュを行わねばならない．しかし，ユーザスタック切り替えの一連の処理コストが高いため，単純にタスク切り替えの都度にユーザスタックを切り替える処理を行うのではなく，実際にユーザスタックの切り替えが必要となる場合のみ実行することで全体のオーバヘッドを低減させる．

具体的には，タスク切り替え時には，切り替え前に動作していたタスクをドメイン毎に記録しておき，タスク切換え時にドメイン切り替えが必要になった際に，切り替え先のドメインで最後に動作していたタスクとタスク切り替え先のタスクが同一のものかどうかを判定する．こうすることで，ページテーブルの書換え回数とそれに伴う TLB フラッシュの回数を低減させることが可能となる．よって，この方式を実現するためには各ユーザドメインに最後に実行したユーザタスクを記憶しておくための領域を用意する必要がある．処理の流れを **図 6** に示す．

6.2 red zone 方式

6.2.1 概　　要

この方式は，ユーザスタックとユーザスタックの間に red zone（1 ページの空間）を作り，この空間を全タスクからアクセス禁止に設定することで，ユーザスタックのスタックオーバ

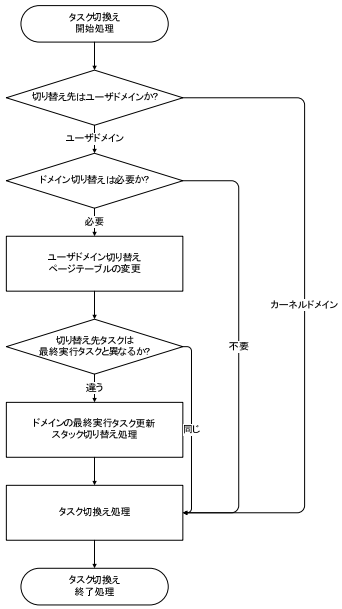

図 6 ページテーブル書換え方式のフローチャート
Fig. 6 the updating pagetable flow chart

フロー・アンダーフローの検出を実現する． **図 7** にタスク切換え時のアクセス許可の変化を示す．タスク 1 を実行中であっても，タスク 2 を実行中であっても，同一ドメイン内であればアクセス可能となっている．しかし，ユーザスタックの間には，red zone が設定してあり，スタックオーバフロー・アンダーフローが発生した場合には，メモリアクセス違反が発生する．

この方式を用いるメリットは，ページテーブル書換え方式と異なりディスパッチャにおいてページテーブルエントリの書換えが不要となることである．その結果タスク切り替え処理を高速化することができる．

しかしその反面，悪意のあるアクセスに対しては無防備であり，もし破壊されてしまったとしても検証することができないということと，red zone のためにタスクスタック毎に 1 ページ分の領域が余分に必要となることというデメリットがある．

ただし，他のドメインのスタックを交互に配置することで red zone の使用数を減少させ

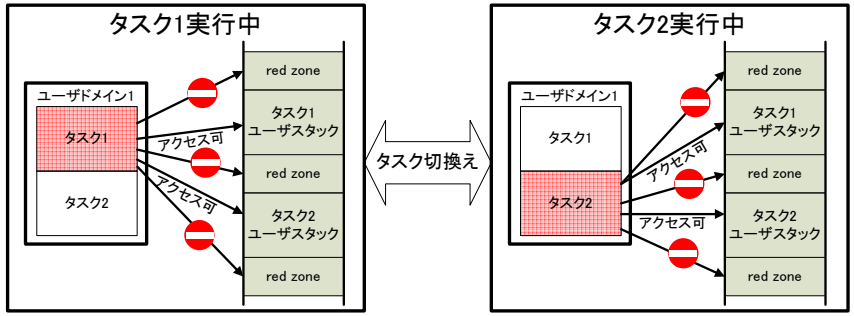

図 7 redzone 方式のユーザスタック保護
Fig. 7 user stack protection using red zone

ることや，アドレス変換を利用してスタックを論理アドレスとして使用し物理アドレスとして存在しない空間を red zone として割り当てることで物理メモリの使用量を削減することで，デメリットを低減させることが可能である．今回の実装では，これらの最適化は見送った．

6.2.2 実　　装

ユーザスタック間を 1 ページ（4KB）の間隔を空けてリンクする．ページテーブル作成時にユーザスタック領域はそれぞれのユーザタスクがアクセス可能となるよう設定するが，スタック間に作成した red zone はページとして変換テーブルを作成しないため，アクセス不能となる．ユーザタスクがこの領域に対して書き込んだ場合には，アクセス禁止領域にアクセスしたとして，データアボートが発生する．発生したデータアボートをハンドラで処理することでスタックオーバフロー・アンダーフローを検出する．

6.3 ARM ドメイン方式

6.3.1 概　　要

この方式は，ARM ドメイン機能を利用して，同一保護ドメイン内で各ユーザスタックに異なる ARM ドメインを割り付けることでユーザスタックの完全な保護を実現する．

タスク切り替えの際に，有効なドメインを切り替えることでアクセスできるユーザスタックを変更する．

この方式を用いるメリットは，ユーザスタックの完全な保護を提供できることに加え，オーバヘッドも red zone を用いた保護と同等程度に抑えられることである．その反面，1 つのユーザドメイン内に生成できるタスクの個数は 15 個までに制限されてしまうこと，論理ア

ドレスと物理メモリを 1 対 1 で利用すると，1MB 単位でしか ARM ドメインを区別できないために物理メモリにフラグメンテーションが発生してしまうこと，ARMv6 アーキテクチャに固有のハードウェアであるため，他のアーキテクチャを持つハードウェアに対して汎用性がないことというデメリットがある．

物理メモリのフラグメンテーションを削減する方法として，red zone の場合と同様に，アドレス変換を利用してスタックを論理アドレスとして使用することが挙げられる．ユーザスタックとして必要な領域のみ物理メモリを実際に割り当て，フラグメンテーションを避けることができる．

また，ASID と ARM ドメインの使い方によって他の実現方法も考えられる．ユーザタスク毎に ASID と変換テーブルを割り当て，ユーザドメインが切り替わる際には ARM ドメインを切り替える方法である．ユーザタスクの text, rodata, data, bss 領域を共有とし，同一ドメイン内で他のタスクとの共有を行う．ユーザスタック領域はユーザスタックを利用するユーザタスクの持つ変換テーブルのみで有効とし，非共有として設定する．他の部分は本論文で述べたものと同様になる．こちらの方法を取った場合には，フラグメンテーションを軽減させることが可能になるが，ユーザドメインの最大数がカーネルドメイン用を除く 15 個に限定されること，TLB の利用効率が低下する可能性があること，本論文内で比較している他の手法との実装上の差異が大きくなりすぎることなどが考えられるため今回は対象としていない．

6.3.2 実　　装

タスク初期化時に各ユーザドメイン内でユニークな ID をタスクに設定する．同一ユーザドメイン内でのタスク切り替え時に，この ID とカーネルが配置されている領域の持つ ID の論理和を ARM ドメイン ID としてレジスタに設定する．こうすることで，各ユーザタスクからは自タスクのユーザスタックはアクセス可能になる一方で，他のユーザタスクの持つユーザスタックにはアクセスすることができなくなる．

7. 評　　価

7.1 実　　験

非特権モードを利用してユーザドメインを動作させるようにしたことで，システムコールの呼び出しが関数呼び出しから，非特権モードから特権モードへと移行するための svc 命令を利用した方式に変更する必要が発生した．そのため，システムコールの呼び出すコストがどの程度増加したかを確認するために，呼び出しから戻るまでの時間を計測し，TOPPERS/ASP カーネルと比較した．

表 3　タスク切り替えのパターン
Table 3　task switching pattern

テスト番号	切り替え元	切り替え先	スタック切り替えあり	ドメイン切り替えあり
1	カーネルタスク	カーネルタスク	×	×
2	ユーザタスク	カーネルタスク	×	×
3	カーネルタスク	ユーザタスク	×	○
4	カーネルタスク	ユーザタスク	○	○
5	カーネルタスク	ユーザタスク	×	×
6	カーネルタスク	ユーザタスク	○	×
7	ユーザタスク	ユーザタスク	×	○
8	ユーザタスク	ユーザタスク	○	○
9	ユーザタスク	ユーザタスク	○	×

また，保護ドメインの切り替えとユーザスタックの変更はタスク切換え時に実行するため，タスク切換えの時間を計測し，TOPPERS/ASP カーネルと比較した．

評価環境としては，ARM11/MPCore を搭載した開発ボードを用いた．コアクロックは 400MHz，キャッシュは命令・データ共に 32KB，TLB は Unified TLB 方式で 64 エントリである．各テストは 10000 回行い，最初の結果は誤差が大きいため捨てている．TOPPERS/ASP カーネルは 1.3.2 を使用した．TOPPERS/ASP カーネルは，メモリ保護機能を持たず，システムコールの呼び出しは関数呼び出しとなっている．

キャッシュや TLB のミスによって計測結果が不安定になることを防ぐため，テスト毎に命令・データキャッシュは全てパージし，TLB も全てフラッシュしている．

システムコール get_tid の呼び出し時間を計測するためのテストをテスト 0 とした．

タスク切り替えのパターンを洗い出したものを **表 3** に示す．タスク切り替えは，切り替え先タスクを rcv_dtq システムコールで待機させておき，切り替え元タスクが snd_dtq システムコール発行後，自らは rcv_dtq で待機状態になることで発生させている．

システムコールの呼び出し時間を計測するテスト 0 と 表 3 のテスト 9 種の合計 10 種のテストを行った．各方式で，TLB ミスの回数が計測結果に影響を与えることが想定できるので，テスト毎に TLB ミスの回数も同時に計測を行った．

テストの結果を，**図 8**，**表 4** に示す．図 8 では横軸にテスト番号，縦軸にテストの実行時間の平均値をとり，計測結果のぶれを標準偏差を使って示している．ページテーブル書換え方式については実行時間のぶれ幅が大きすぎたので，グラフから突き抜けてしまっている．

Vol.2009-EMB-14 No.8
2009/7/24

7.2 考　　察

図 8 から，システムコールの呼び出しオーバヘッドを計測するためのテスト 0 の結果を見ると，システムコールの呼び出しを変更したことによって TOPPERS/ASP カーネルよりも 163 サイクル余分にかかってはいるものの，TOPPERS/HRP2 カーネル内では方式による差異がないことが分かる．

また，他のテストを見るとページテーブル書換え方式のテスト 4，6，8，9 の実行時間が他と比較して高いことが分かる． 表 3 を参照すると，テスト 4，6，8，9 は全てユーザスタックの切り替え処理を含んでいる．そこで， 表 4 を参照すると，テスト 4，6，8，9 の TLB ミスの回数は他と比較すると大きくなっている．これらのことから，ユーザスタックの切り替え処理時に TLB フラッシュを実行し，それが原因となって TLB ミスが増加していると考えられる．また，実行時間のぶれもテスト 4，6，8，9 では大きくなっている．例として，TOPPERS/ASP カーネルとページテーブル書換え方式のテスト 8 における実行時間の度数分布を **図 9** に示す． 図 9 では横軸に実行時間をとり，縦軸に発生頻度を対数でとっている． 図 9 から，TOPPERS/ASP カーネルの度数分布を見ると，9999 回の試行中 9998 回の実行時間が 342 サイクルとなっているのに対し，ページテーブル書換え方式ではばらつきが大きく収束していないことが見て取れる．このため，ページテーブル書換え方式ではリアルタイム性が損なわれてしまっているといえる．

red zone 方式の計測結果には大きなぶれはなく，TOPPERS/ASP カーネルでの計測と同様にテスト毎の大幅な性能変化も見られなかった．よって，これらの方式はユーザスタックのサイズ，タスク切換え時の処理内容によらず，リアルタイム性を維持しているといえる．他の方式と比較すると，red zone 方式はスタックオーバフロー・アンダーフローにのみ対応できるという点で保護機能は完全ではないが，他のドメインからのアクセスに対しては，ページテーブル書換え方式と同様に完全な保護を提供することができる．よって，同一ドメイン内での保護を重視しない場合であれば問題なく利用することが可能である．ただし，red zone を構築するためにメモリの使用量が実際に使用する量に加えて余分に必要となるため，リソース制限の厳しいシステムで利用する場合には工夫が必要となる．

ARM ドメイン方式については，red zone 方式と同様にリアルタイム性を維持しているが，テスト 2 および 8 の結果が悪化している．現時点で原因は調査中であり，よく分かっていない．ARM ドメイン方式は完全な保護を提供することができるが，TLB の動作などの非公開情報に頼る部分があり，不確定要素が大きいため用いる場合には注意が必要である．

図 8 と 表 4 を比較すると，ページテーブル書換え方式のテスト 4，6，8，9 は TLB ミスが多いことがオーバヘッドに影響を与えていることが見て取れる．また，テスト 2 では，red zone 方式と ARM ドメイン方式を比較すると，TLB ミスの回数が約 2 倍になるのに従って実行時間も約 2 倍に延びている．しかし，テスト 3 では，TLB ミスの回数が約 1.5 倍に増えているのにも関わらず，実行時間はほぼ同一のものとなっている．このことから，TLB ミスのみが実行時間に影響を与えているわけではないことが考えられる．

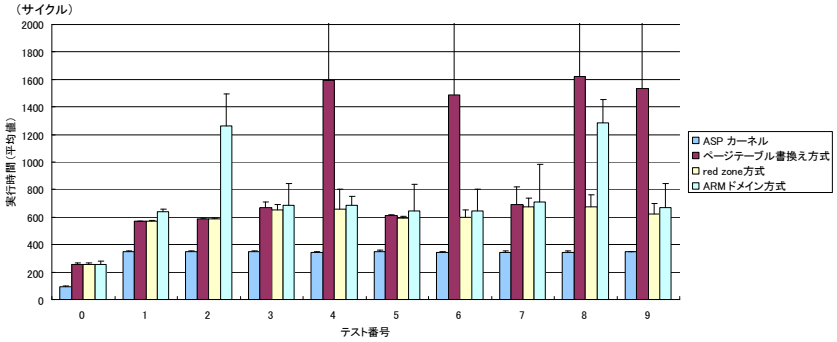

図 8　オーバヘッド計測結果
Fig. 8　the result of overhead measurement

評価の結果を踏まえて，各方式の利点と欠点を比較したものを **表 5** に示す．ページテーブル書換え方式は，必要なメモリ量を抑えることができ，ユーザスタックの完全な保護を行うことができる反面，タスク切り替えオーバヘッドのぶれが大きいため予測可能性が低くリアルタイム性に欠ける．red zone 方式は，低いタスク切換えオーバヘッドで実現でき，ユーザスタックのスタックオーバフロー・アンダーフローを検出することができる．その反面，利用できないメモリ空間が発生するためメモリの利用効率が低下することと，悪意のあるアクセスに対しては無防備であることが欠点となる．ARM ドメイン方式は，ユーザスタックの完全な保護を行うことができ，低いタスク切換えオーバヘッドで実現できる．その反面，保護を 1MB 単位で行わなければならず必要メモリ量が増大することが欠点となる．

今回は見送ったが，ページテーブルを静的に生成することができるならば，ページテーブル書替え方式以外の場合には，ROM 領域にページテーブルを配置することができ，リソース制約の厳しい組込みシステムにも適用することが可能である．

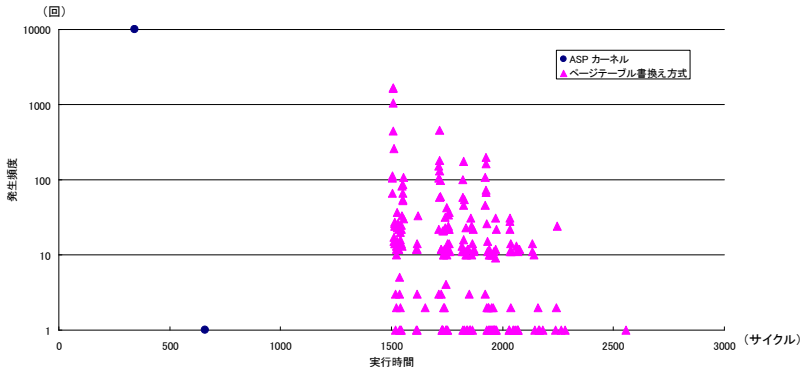

図 9 実行時間のぶれ
Fig. 9 the jitter of ASP kernel and the updating pagetable

表 4 TLB ミス回数計測結果
Table 4 the result of tlb miss measurement

テスト番号	0	1	2	3	4	5	6	7	8	9
ASP カーネル	0	0	0	0	0	0	0	0	0	0
ページテーブル書換え方式	21	21	23	45	38334	23	25351	55	38829	30945
red zone 方式	21	20	22	45	55	22	34	45	45	34
ARM ドメイン方式	21	26	44	62	79	44	53	62	79	53

表 5 方式毎の特徴比較
Table 5 comparing each method

	ページテーブル書換え方式	red zone 方式	ARM ドメイン方式
タスク切り替えオーバヘッド	×	○	○
必要メモリ量	○	△	×
保護の完全性	○	△	○

8. ま と め

本研究では，ARMv6 アーキテクチャを用いたメモリ保護 RTOS のユーザスタック保護の設計と評価を行った．本研究で設計したユーザスタック保護方式は，ページテーブル書換え方式と red zone 方式と ARM ドメイン方式である．

これらの方式には全て長所と短所があり，システムの設計や信頼性要件によって使い分けることが必要である．メモリ保護機能は今後組み込みシステム開発の重要な機能になっていくことが予想される．提案手法を使うことで既存のアプリケーションに大幅な変更を与えることなくメモリ保護機能を提供でき，システム開発の一助となることが期待される．

今後の課題として，事前に変換テーブルを構築できるようにすることが挙げられる．変換テーブルを構築するには，全てをリンクした状態でのアドレス情報が必要であるため，現在は実行時に構築するようになっている．red zone 方式と ARM ドメイン方式の場合実行中にページテーブルを書き換える必要がないため，事前に変換テーブルを構築しておくことで変換テーブルを ROM 領域に配置することが可能となり，RAM 領域の使用量を削減することが可能である．

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に厚く御礼申し上げます．

参 考 文 献

1) 西部 満，本田晋也，冨山宏之，高田広章：ハードリアルタイムシステムに適したメモリ保護機構の提案と評価，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.104, No.738, pp.25–30 (2005).
2) ARM: ARM Architecture Reference Manual (online) , available from (https://www.jp.arm.com/document/manual/files/051020DDI0100HJ_v6_1.pdf) (2005).
3) the Linux Kernel Organization, Inc: The Linux Kernel Archives (online), available from (http://kernel.org/).
4) On-Line Applications Research Corporation: RTEMS C User's Guide (online) , available from (http://www.rtems.com/onlinedocs/releases/rtemsdocs-4.9.2/share/rtems/html/c_user/index.html) (2009).
5) AUTOSAR GbR: Specification of Operating System (online) , available from (http://www.autosar.org/download/AUTOSAR_SWS_OS.pdf) (2009).