2007－EMB－6 (3)
2007／12／4

SMP-Accounting System: SMP 資源管理手法の提案

菅谷 みどり† 湯浅 陽一†† 中島 達夫†

近年，組込みシステムの高度化する機能要求を満たす為，マルチコアプロセッサが普及してきた．マルチコアの中でも特に SMP(Symmetric Multi Processor) は OS 間通信のオーバーヘッドが無く，プロセッサ固定化機能によるリアルタイム性能の改善等により，組込み分野での適用領域が拡大している．しかしながら，プロセッサ固定化機能を用いたシステム設計では，プロセッサ数が変更する場合に，ユーザプログラム側の書換えが必要となり，その再利用性が低下する問題があった．本論文では，SMP 環境下でのユーザプログラム開発の負担を軽減するため SMP-Accounting System というミドルウェアを提案する．本システムは，プロセッサ論理番号やスケジューリングパラメータを抽象化したインターフェイスと制御機能を提供する事でユーザプログラムの負担を軽減する．本稿ではその有効性について論じる．

SMP-Accounting System: Resource Management System for Multiprocesor

Midori Sugaya,† Yoichi Yuasa†† and Tatsuo Nakajima†

Recently, to satisfy increasing requirements of advanced embedded system with multicore become popular. In multicore system, SMP (Symmetric Multi Processor) is preferred by less overhead of communication between operating systems and less cost of collaboration of applications and realize real-time performance with processor affinity function. However, the cost of complexity of applications is also high in the SMP system. In this paper, we propose a SMP-Accounting System that can decrease the cost of building embedded system with satisfying the requirements in SMP by using it's abstract model.

1. はじめに

近年，組込みシステムでは，高度化するソフトウェアやサービスの機能要求を満たす為，マルチコアをベースとした高性能のプロセッサが普及している．マルチコアプロセッサは，プロセッサの並列化による消費電力の低減等の理由により組込み分野での適用領域が拡大している．

マルチコアの中でも SMP(Symmetric Multi Processor) は OS 内のリソース共有が可能である事や，AMP(Asymmetric Multi Processor) に比べ OS 間通信のオーバーヘッドが無い等の理由により注目が高まっている．しかし従来は，SMP を実現する Linux にみられるように，ロードバランシング機能によるタスクの実行順序の予測が困難である等の問題があった．これに対し最近，SMP-Linux でも特定のプロセッサにタスクを固定化するアフィニティ[2] 機能により実行順序の予測性の問題改善がなされ，実用的な組込みシステムへの適用への期待が高まっている．

しかし，プロセッサの固定化機能では，物理プロセッサの ID をアプリケーションが管理する必要がある．その為，プロセッサの個数が変更する度に，アプリケーションの書換えが必要となる．同様に，プロセッサ資源割り当ての調整が必要となる．これらの書換えや調整をアプリケーション側で行う事は，その再利用性を低下させるという問題がある．

本研究では，アプリケーションの再利用性を向上させる手法として SMP-Accounting System を提供する．SMP-Accounting System は，従来アプリケーション側で管理していたプロセッサ ID とスケジューリングパラメータ等をアプリケーションに代り管理，及び制御機能を提供するするミドルウェアである．導入により，アプリケーションの再利用性を高めることができる．本論文では SMP-Accounting System の設計・実装，及び有効性について述べた．

本論文の構成は，第 2 章にて，組込みシステムの現

† 早稲田大学理工学術院
Waseda University, Department of Computer Science
†† トライピークス株式会社
TriPeaks Corporation

状と SMP システムの課題を述べ，第 3 章にて SMP-Accounting System を提案する．第 4 章にて実装を述べ，第 5 章で結論を述べる．

2. マルチコアシステムの現状と課題

2.1 背　　景

近年の組み込みシステム開発では，高度化する機能要求が顕著である．例えば，動画・音声、ネットワークなど PC 並の機能が組み込み機器にも組み込れるようになった．

こうした機能要求を満たす為に，プロセッサも特定処理に特化したものではなく，汎用プロセッサを用いる事で多様なアプリケーションを動作させ，全体のコスト削減を目指す動きがみられる．さらに，低消費電力への要求により，プロセッサの並列化による電力消費量の低減を目的としたマルチコアの採用が進んでいる．一方で，競争激化による開発期間の短縮により，アプリケーション資産の再利用性がより重要な課題となっている．

2.2 SMP の利点と問題点

こうした中，マルチコアの中でも特に SMP は次の利点から組込み分野での適用領域が拡大している．(1) 並行処理による低消費電力化，応答性の改善．(2) 単独の OS を利用できるため，OS 内のリソースを共有が可能である．さらに，資源を効率よく利用する為の仕組みとして，ロードバランス機能等が提供されている．(3) AMP のようにプロセッサ毎に異なる OS を乗せる必要が無い為，OS 間通信のオーバーヘッドが無い．さらに (4) プログラムをマルチスレッド化する事によりプロセッサの並行処理時の性能を最大限利用できる事などがあげられる．

これに対して，SMP の問題点としては (1) 処理性能の向上には，プログラムのマルチスレッド化が求められる．また，(2) 利点であるロードバランシング機能は，アプリケーションがどのプロセッサで実行されるか予測が難しい事から，リアルタイム性の保証の観点ではデメリットとなる点があげられる．

2.2.1 SMP での問題への対処

問題点の (1) については，既に POSIX 等のスレッドライブラリなどのミドルウェアが提供されている．問題点の (2) に関しては，最近では，SMP 向けの OS を提供する Linux では，プロセッサの固定化の為の手法（プロセッサアフィニティ[2)]）が提供され，プロセッサの固定化及び優先度管理によるリアルタイム設計が可能となっている．

しかしながら，プロセッサの固定化は，アプリケーションごとにプロセッサの物理 ID やスケジューラビリティを管理する必要がある．その為，プロセッサ数の変更があった場合，その都度依存するアプリケーションの書換えが必要となる．将来的な SMP 環境では，省電力化を目的とした CPU のホットスワップなど，プロセッサ数の動的な変更が想定される為，アプリケーションの書換えがその都度必要となる現状の仕組みでは，手間が増大する問題がある．問題に対処するには，アプリケーションの再利用性を考慮した仕組みが必要である．

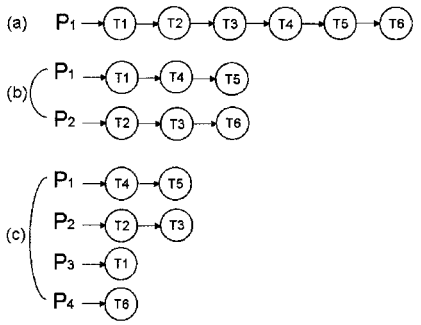

図 1 プロセッサ数の変更 (SMP-AS 無)
Fig. 1 Change the number of processors (w/o SMP-AS)

仮定：移動先プロセッサ P_m，移動するタスク τ_j は事前に決定．
例 (a) $m = 1$, $\{\tau_j | j = 1, 2, \ldots, 6\}$
例 (b) $m = 2$, $\{\tau_j | j = 2, 3, 6\}$
例 (c) $m = \{3, 4\}$, $\{\tau_j | j = 1, 6\}$

```
1:  cpu_set_t mask;
2:  pid_t pid = τj;
3:  int cpu_id = Pm;
4:
5:  /* clear all bits in the mask; */
6:      CPU_ZERO (&mask);
7:
8:  /* set a new processor id for the mask */
9:      CPU_SET (cpu_id, &mask);
10:
11: /* assign a task to a processor */
12: if (sched_setaffinity (pid, sizeof(mask), &mask) == -1)
13:     printf ("Failed to set CPU affinity");
```

図 2 τ_2 の P_2 への移動時のコード例 (w/o SMP-AS)
Fig. 2 Example of the change of processor binding w/o SMP-AS

3. SMP-Accounting System

3.1 提　　案

2.2.1 節で述べたように，タスクをプロセッサに固定化する事により，実行の予測性は向上する．但し，アプリケーションの書換えの手間 (コスト) は増大する．具体的なコストは，

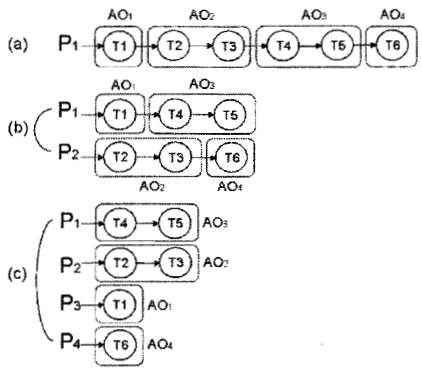

図 3 プロセッサ数の変更 (SMP-AS 有)
Fig. 3 Change the number of processors (w/SMP-AS)

- プロセッサ ID の管理
- 複数プロセッサのスケジューラビリティ管理

に関する追加的な開発（設計・実装）コストである．

これらのコストを低減する為に，本研究では OS 内のミドルウェア SMP-Accounting System (以降 SMP-AS) を提案するものとした．SMP-AS は従来アプリケーション側でが管理していたこれら管理を，アプリケーションに代わり行うための機能及び制御システムを提供する．以降，アプリケーションプログラムについてはプログラム，実行単位をタスクと呼ぶ．

3.2 機　　能

本研究では，プログラムの書換えに伴うコストを低減する手段として SMP-AS が持つ三つの機能を定義した．

グルーピング機能: SMP-AS では，タスクをグループ化して扱う論理インターフェイスとなるオブジェクトを，アカウンティングオブジェクト (Accounting Object: 以降 AO) と定義した．また，AO には複数タスクを関連付け（バインド）できるものとした．グループ化のメリットは，タスク数の増大に対して，タスクを扱う手間が単純に増大しないモデルを作成する事ができることである．

AO による操作パラメータ保持機能: AO が保持するパラメータには，AO に所属するタスクと，操作を規定するパラメータ（固定化するプロセッサ ID，スケジューリングパラメータ等）がある．AO は，自身に所属するタスクと，タスク操作のパラメータを保持することで，タスクと操作の記述の間に介在し，それらの関係を定義する機能を持つ．

制御機能: SMP-AS では，パラメータに基づき制御を行うための機能を提供する．制御機構はカーネルのサービスを利用する必要があるため，ミドルウェアとして設計した．

仮定：(共通) プロセッサ: $P_m, m > 0$

1. AO の作成とタスクとの関係定義:
(a) $m = 1$, $\{\tau_j | j = 1, 2, ..., 6\}$

```
1:  pid_t pid = τj;
2:  account_t object_attr;
3:
4:  /* set processor id for the object attribute */
5:      object_attr → cpu_id =Pm;
6:
7:  /* create accounting object */
8:      object_id = create_account(&object_attr);
9:
10: /* bind task for the accounting object */
11:     bind_pid(object_id,pid);
```

2. AO とプロセッサの定義：
図 1 例 (b) 移動 AO_i,$\{AO_i | i = 2, 4\}$,$m = 2$

```
1:  AO2: object_attr → cpu_id =P2;
2:      set_account (&object_attr);
3:  AO4: object_attr → cpu_id =P2;
4:      set_account (&object_attr);
```

図 1 例 (c) 移動 AO_i,$\{AO_i | i = 1, 4\}$,$m = \{3, 4\}$

```
1:  AO1: object_attr → cpu_id =P3;
2:      set_account (&object_attr);
3:  AO4: object_attr → cpu_id =P4;
4:      set_account (&object_attr);
```

図 4 タスクのプロセッサ移動時のコード例 (b)(c)
Fig. 4 Example of the change of processor binding(b)(c)

3.3 プログラミングの分離モデル

通常プログラムには，プログラムの実行主体が行うべき操作を記述する．しかし，本モデルではタスクと操作との関係を AO を通じて分離する．さらにタスクの操作の一部を SMP-AS システム側で行う．

例えば，プロセッサ ID の固定に際して，通常タスク側のプログラムでは，タスク自身が固定されるべきプロセッサ ID を記述する．これに対して本モデルでは，タスクはプログラムに AO との関係の定義を行う記述を行い，ミドルウェア側では，AO とプロセッサ ID の関係を定義する記述を行う．これはつまりタスクとプロセッサ ID の関係定義を，

- タスクと AO の関係の定義
- AO とプロセッサ ID の関係の定義

の 2 つに分離している．その為，プロセッサ構成が変更した場合であっても，タスク側での再定義が不要となる．代わりに，AO 側で再定義を行う．複数タスクの定義を統一的に変更するといった拡張も SMP-AS

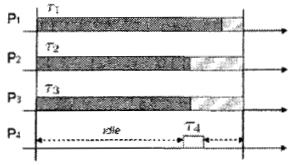

図 5 従来の SMP スケジューリングの例
Fig. 5 Conventional SMP scheduling

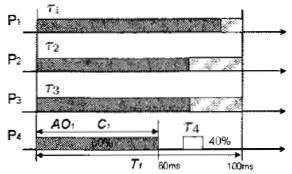

図 6 SMP-AS を用いたスケジューリング例
Fig. 6 w/ SMP-AS scheduling

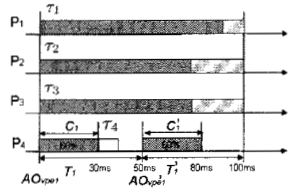

図 7 VPE を用いたスケジューリング例
Fig. 7 w VPE on SMP-AS

- τ_i : i 番目のタスク
- P_i : i 番目のプロセッサ
- AO_i : i 番目の AO (Accounting Object)
- C_i : AO_i の計算時間
- T_i : AO_i の周期

図 8 記号一覧

内で容易に実現できるため，プログラミングの拡張性も高い．このように，AO を介在してタスクと操作の関係を分離し，SMP-AS 側で定義や操作を代替する事で，タスク側のプログラミングをシンプルにすると同時に，拡張性が高いシステムを構築することができる．

3.4 プログラミング例

本節では，プロセッサ ID の変更時に必要となるプログラミングの例を，通常の場合（図 1）と，SMP-AS を用いた場合（図 3）の対比で示した．本論文で用いる記号を図 8 に示した．

図 1 では，(a) の P_1 上のタスクの一部をそれぞれ (b) P_2, (c) P_2, P_3, P_4 に移動させる例を示している．タスク $\tau_j\{j|1,2,...,6\}$ のうち，(τ_2,τ_3),(τ_4,τ_5) を依存関係のあるタスクとし，同じプロセッサ上で動かすことを前提とする．図 2 に τ_j を P_1 に固定化するコードを示した．*sched_setaffinity* は，プロセッサ ID のビットマスクを引数に取り，呼び出されたタスクを P_m に固定化するシステムコールである．(a) では同様のコードが τ_1 から，τ_6 までのタスクに必要である．(b),(c) では，事前に依存関係を考慮の上，移動タスクを決定する．(b)τ_2, τ_3, τ_6, (c) τ_1, τ_6 のコードの書換えがそれぞれ必要となる．

これに対し，図 3 は，AO を用いた例を示した．SMP-AS では，はじめに AO とタスクの定義を行う (図 4 上)．タスクの依存関係は，図 1 の例と同様だが，AO の属性 (object_attr) のパラメータとしてプロセッサ ID を指定する（5 行目）．object_attr を引数にして，システムコール *create_account* により AO を作成する（8 行目）．戻り値に AO の ID が返されるので，その ID に次はバインドすべきプロセスを指定する（11 行目）．元に新しい AO 単位でタスクをグループ化する (4-6 行目). 図 3 の (b),(c) にて，プロセッサ構成の変更時のコードを示した．変更は AO とプロセッサの再定義となる．AO の属性にあるプロセッサ ID のパラメータを書き換えて（1 行目), *set_account* を実行することで，AO は指定されたプロセッサにタスクを移動させる．このように，パラメータの設定をタスク側のプログラムで行う方法から，SMP-AS 側で行うようにモデルを分離する事により，タスク側のプログラムがシンプルになる．

3.5 スケジューリングパラメータの管理

本節では，SMP-AS を用いて，AO にスケジューリングパラメータを設定する事で，より容易にプロセッサのスケジューラビリティの管理が行える事を示す．今回本研究の基盤とした Accounting-System[1] は，スケジューリングの制御パラメータとして，実行時間 (C) と周期 (T) の二つの値を AO ごとに設定し，AO に属するタスクをそのパラメータにそって制御する機能を提供している．これら二つの値は，周期とデッドラインを同一と仮定した場合の最も単純な周期タスク制御のモデルを構成するパラメータであり，プログラマが直感的な資源配分を設定する事が可能である．SMP-AS ではプロセッサ ID 情報を AO に追加した．次に，スケジューリングの制御パラメータを用いた SMP の資源管理の有効性について述べる．

3.5.1 従来の SMP スケジューリング

SMP を用いる高機能な組込みシステムでは，システムに対するサービス要求も高い．例えば，音声や画像処理と同時に GUI 操作の応答性の両方を同時に満たす事が求められる．従来，汎用機等の SMP システムでは，前者に対して優先的にプロセッサを割りあて，後者の周期的ではないが応答性が必要なタスクに，残りのプロセッサのうち一つを専用で割り当てる手法が一般的にとられてきた．図 5 に例を示す．プロセッサ $P_m = \{1,2,3,4\}$，タスク $\tau_j = \{1,2,3,4\}$ のうち，τ_1- τ_3 は周期タスクとし，P_1 -P_3 上でそれぞれ動作させるものとする．τ_4 は非周期タスクとし，P_4 に割

り当てる．

図5に示した方法の利点は，容易に周期タスクのリアルタイム性能と非周期タスクの応答性を両立できる点と，周期タスクが過負荷になっても，プロセッサに固定されていればロードバランスされず，非周期タスクが常に応答性を確保できる点である．但し，図5からも見取れるように，非周期タスクは待ち時間が長い為，専用で割り当てたプロセッサ (P_4) の利用率が低い．組込みシステムでは，SMP を利用する場合であっても，資源の無駄はコストの観点からも許容されない為，改善が必要である．

3.5.2 SMP-Accounting System の利用

SMP-AS により，P_4 の CPU 利用率を向上させる設計が可能である．図6に例を示した．例えば，τ_4 は周期 100ms のうち，最大 40ms を利用するタスクである場合，残りの時間は，別のタスクに割り当てる事ができる．この場合，P_4 上に AO_1 を作成し，プロセッサ ID ($m = 4$), スケジューリングパラメータ ($C_1 = 60ms, T_1 = 100ms$) を設定し，周期タスク τ_5 をバインドする．τ_5 は，100ms 周期のうち 60ms を利用し，それ以外の 40% を非周期タスク τ_4 が利用する事ができる．AO_1 には，τ_5 以外の複数タスクをバインドする事も可能である．

このように，SMP-AS によりプロセッサの利用率の操作を行う事ができる．但し，AO に設定された周期に対する実行時間比率が高い場合，非周期タスクの応答性が低下する問題がある．

非周期タスクの応答性の問題については，我々は本研究プロジェクトの一部として VPE (Virtual Periodic Execution)[3] を提案した．VPE は，周期タスクの周期 (T) と実行時間 (C) を分割係数で分割する事により，非周期タスクの応答性を向上させる手法である．VPE による制御の例を図7に示した．図では，分割係数を 2 とした場合，$C_1 = (C_1[30ms], C_1'[30ms])$, $T_1 = (T_1[50ms], T_1'[50ms])$ により，τ_4 の最悪応答時間が 60ms から 30ms へと 1/2 倍された結果を示している．

また，本手法以外にも，非周期タスクの応答性を高める研究は数多く提案されている[4][5]．SMP-AS では，多様なアルゴリズムをポリシーとして利用できる．その為，システム環境に合わせてポリシーを選択する事で，リアルタイム性，非周期タスクの応答性，CPU 利用率の向上等の目的をより容易に満たすことができる．

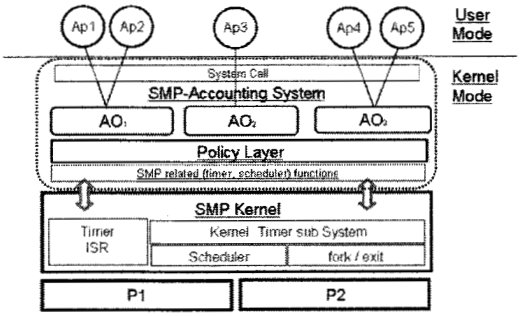

図 9 SMP-AS アーキテクチャ
Fig. 9 SMP-AS Architecture

4. 実　　装

4.1 全体構成

本開発では，基盤として Accounting System[1] を用いた．Accounting System は，カーネル内にタスク制御に必要な最小限のフックを設置し（起動，割り込み，スケジューリング，fork, exit) タスクの動作を制御する仕組みとなっている．

SMP-AS の実装は，Linux の 2.6.18 の上に行った．SMP-AS はカーネルモードで動作するミドルウェアであり，カーネルに静的にリンクされる．今回，Accounting System が提供する AO の操作，制御機能は,SMP プロセッサ対応に変更を行った．特にプロセッサごとの固定機能，時間管理機能，タスクキューはシングルプロセッサと SMP は異なる為，対応する仕組みを開発した．又ロックの書換えなどにより，RT Preemption 対応を行った．開発の基盤とした Accounting-System は，cabi-v1.1-x86 シングルプロセッサベースの実装を用いた．図9に全体構成を示す．

プロセッサの固定化: Linux では，2.6 バージョンのカーネルでは標準でプロセッサアフィニティのシステムコールが実装されており，SMP-AS でも，内部的にこのカーネルのアフィニティを用いた．プロセッサアフィニティ機能は，タスクが新たに生成される時に，タスク構造体の cpu_allowed メンバにマスクを設定する．スケジューラはスケジューリング時にマスクをチェックし，マスクと一致したプロセッサのみでプロセスを実行する仕組みとなっており，ロードバランシング機能の影響を受けない．

時間制御: シングルプロセッサ用のアカウンティングシステムの拡張を行った．AO 毎に属するタスクの実行時間 (C) を周期的にモニタリングし，実行時間が設定時間を越えた場合に，定められた動作（シグナ

ルを管理プロセスに送る，タスクをブロックする）を実施する．

4.2 API

現在 SMP-AS は，システムコールによりプリミティブである API を実装しているため，引数に値を設定した上で，API を呼び出す手順となっている．

&object_attr は，AO の ID，プロセッサ ID，スケジューリングパラメータ（周期，実行時間），実行時間に達した場合の動作，スケジューリングポリシーをメンバ値として持つ構造体である．ユーザはプログラムの中で，この構造体のメンバ値を要求に従って設定する．作成時には AO の ID を構造体に指定せずにシステムコールを呼び出し，その戻り値に AO の ID が設定されて返される．次に API リストを示す．

- **account_create(&object_attr)**：AO の作成
- **account_destroy(&object_id)**：AO の削除
- **bind_pid(object_id, pid_t)**：AO へのタスクのバインド
- **unbind_pid(pid_t)**：タスクのアンバインド
- **account_{get/set}(&object_attr)**：AO に設定されている値の取得/設定

account_create の API は *&object_attr* の構造体のアドレスをユーザから受け取り，カーネルメモリ領域に構造体をコピーして保持する．account_destroy はカーネル内の AO のオブジェクト領域の削除を行う．AO へのタスクの登録/削除はそれぞれ bind_pid, unbind_pid，AO への値の取得や変更は account_get, account_set の API により行う．

4.3 拡張性の考慮

本 SMP-AS では，タスクのスケジューリング，プロセッサの配分についてのポリシーを容易に拡張する事ができる仕組みを提供している．スケジューリング時の振る舞いをメソッドとして定義する *account_operations* 構造体を定義した（図 10 上）．周期の終了時の処理を *replenish*，実行時間の終了時の制御を *enforce* にコールバック関数として周期処理，実行時の処理のアルゴリズムを追加可能とした（図 10 下）．

実際に，本システムによる制御が開始すると，account_operation はそのタイプに適切なメソッドを呼び出す事で異なるポリシーを利用できる．本方式により開発者側は容易に新しいアルゴリズムを本システムに追加する事が可能となる．同時に，ユーザ側がシステム構成や動的な環境の変化にあった適切なポリシーを選択する事ができる．また，操作は実行時に行う事ができる為，環境の変化にそって動的にポリシーを切り替えるシステムの構築も可能である．

```
/* definition of the account operation structure */
struct account_operations {
void (*replenish)(cabi_account_t);
void (*enforce) (cabi_account_t,struct task_struct *);
}
/* this is the cyclic executive policy    */
/* that register their operation.         */
struct cabi_account_operations cyclic_ops = {
.replenish = cyc_replenish,
.enforce = cyc_enforce,
};
```

図 10 関数テーブル
Fig. 10 Structer of account_operations

5. 結　　論

本論文では，組込みでの SMP システムの普及について発生するプログラミングの複雑さを低減する為のミドルウェアを提案した．SMP-AS は，アプリケーションと操作を AO により分離し，AO と一部の操作をミドルウェア側で負担する事により，アプリケーションのプログラミングをシンプルにするた．例では，プロセッサ構成の変化時に，アプリケーションとプロセス ID を再定義せずに，プロセッサの配分の変更を行えることを示した．今後より動的な環境変化に対応するミドルウェアとしての評価を発展させ，さらに有効性を示す事を目標とする．

参 考 文 献

1) Midori Sugaya, Shuichi Oikawa, Tatsuo Nakajima: Accounting System: A Fine-Grained CPU Resource Protection Mechanism for Embedded System. ISORC 2006: 72-84.
2) http://www.kernel.org/pub/linux/kernel /people/rml/cpu-affinity/
3) Midori Sugaya and Yuki Kinebuchi,Shuichi Oikawa, Tatsuo Nakajima, VPE: Virtual Periodic Execution for Embedded System, 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2007), Work-in-Progress Session.
4) B. Sprunt, Aperiodic Task Scheduling for Real-Time Systems, Ph.D. thesis, Dep. of Electrical and Computer Engineering, Carnegie Mellon University (1990).
5) Lehoczky, J. P.; L. Sha; and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard RealTime Environments. ", In Proceedings of RTSS (Phoenix, AZ, Dec. 2-4).IEEE Computer Society Press, Los Alamitos, CA, 110-123. 1992.