Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

# タスクのデッドラインのみを用いる時間保護スケジューリングアルゴリズム

松原 豊[†1] 本田晋也[†1] 高田広章[†1,†2]

個別に開発・動作検証されたリアルタイムアプリケーションを，単一のプロセッサに統合して動作させるための階層型スケジューリングが数多く提案されている．本論文では，タスクのデッドラインの情報のみを用いる従来手法をベースに，アプリケーション内のタスクをスケジュールするローカルスケジューリングアルゴリズムとして，固定優先度ベーススケジューリングを用いる場合でも時間保護を実現できるアルゴリズムを提案する．さらに，統合前にスケジュール可能なタスクは，提案アルゴリズムにより，統合後にもスケジュール可能であることを示す．

# Hierarchical Scheduling with only task's deadlines for Temporal Protection

Yutaka Matsubara,[†1] Shinya Honda[†1]
and Hiroaki Takada [†1]

Many hierarchical scheduling algorithms have been studied for integrating multiple independently developed applications. In this paper, we present a new scheduling algorithms based on BSS (Bandwidth Sharing Server) that uses only task's deadlines in runtime. It is proven that the proposed algorithm guarantees that a task, which is schedulable on a dedicated processor, is schedulable by using fixed priority scheduling as a local scheduling on a shared processor.

†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 Systems, Nagoya University
†2 名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Engineering, Graduate School of Information Science, Nagoya University

## 1. はじめに

自動車制御システムに代表される分散リアルタイムシステムの開発において，個別に開発・動作検証されたリアルタイムアプリケーションを，単一のプロセッサに統合して動作させるための階層型スケジューリングが数多く提案されている．我々は，容易にアプリケーション統合を実現するためには，個別に動作する場合にスケジュール可能であることを検証したアプリケーションが，統合した後にもスケジュール可能であることを保証できるスケジューリングアルゴリズム（これを時間保護スケジューリングアルゴリズムと呼ぶ）を採用することが望ましいと考える．

これまで，タスクの起動時刻と最悪実行時間を用いるアルゴリズム[1)] や，タスクの起動時刻と相対デッドラインを用いるアルゴリズムが提案されている[2)]．これらは，実行時に，タスクの起動時刻の情報が必要であるため，適用できるアプリケーションが限定される．また，アプリケーションの設計時に，タスクの起動周期（もしくは，最小到着間隔）と最悪実行時間を用いてアプリケーションの起動周期を決定し，実行時は，すべてのアプリケーションを周期的に起動するアルゴリズムも提案されている[6)]．これらのアルゴリズムは，時間保護を実現できることに加えて，実行時の処理オーバヘッドを少なくできるという利点があるが，その一方で，アプリケーションの起動周期を決定するためには，設計時にタスクの詳細な情報が必要であることと，アプリケーションに割り当てるべきプロセッサ利用率が，先の手法に比べて高くなるという欠点がある．

従来手法のうち，タスクのデッドラインのみを用いる階層型スケジューリングアルゴリズム[3),4)] は，タスクの起動時刻や最悪実行時間の情報を用いるアルゴリズムに比べて，適用できるアプリケーションが多いという利点をもつ．しかし，アプリケーション内のタスクをスケジュールするローカルスケジューリングアルゴリズムとして，固定優先度ベーススケジューリングを用いる場合，個別に動作する際にはスケジュール可能であるタスクが，統合した後にデッドラインをミスしてしまう問題がある[5)]．

本論文では，タスクのデッドラインの情報のみを用いる従来手法を拡張することで，時間保護を実現できる階層型スケジューリングを提案する．具体的には，まず，同一のアプリケーションに属する，デッドラインの長い高優先度タスクが，デッドラインの短い低優先度タスクの実行時間を奪うことが，デッドラインミスの原因となり得ることを指摘する．そして，デッドラインの長い高優先度タスクの起動時刻を遅延させることで，低優先度タスクがデッドラインをミスすることを防ぐようアルゴリズムを拡張する．最後に，提案アルゴリズ



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

ムにより，統合前にスケジュール可能なタスクは，統合後にもスケジュール可能であることを示す．

以下，本論文の構成を示す．第 2 章では，本研究が対象としているシステムと，従来手法の問題点を具体的に述べる．第 3 章では，提案するスケジューリングアルゴリズムについて述べる．第 4 章では，提案アルゴリズムにより，統合後にタスクがデッドラインを満たせることを示す．最後に，第 5 章で，結論と今後の課題を述べる．

## 2. システムモデルと従来手法の問題点

### 2.1 想定するシステム

本研究では，分散リアルタイムシステムにおいて，ネットワークを構成するノードのコンピュータシステム（これを個別プロセッサと呼ぶ）上でデッドラインを満たして動作するリアルタイムアプリケーションを，高性能なコンピュータシステム（これを統合プロセッサと呼ぶ）に統合して動作させることを想定している．このとき，統合前に性能 U の個別プロセッサでスケジュール可能なアプリケーションのタスクは，統合プロセッサで，プロセッサ利用率 U を割り当てれば，スケジュール可能であることを保証する階層型スケジューリングアルゴリズムを考案することを目的としている．

階層型スケジューリングアルゴリズムは，アプリケーション内のタスクをスケジュールするスケジューラ（これをローカルスケジューラと呼ぶ）と，どのアプリケーションのタスクをプロセッサで実行するかを決定するスケジューラ（これをグローバルスケジューラと呼ぶ）を 2 階層に配置したアルゴリズムである．

### 2.2 BSS アルゴリズム

本論文では，タスクのデッドラインの情報のみを用いる階層型スケジューリングアルゴリズムである BSS（Bandwidth Sharing Server）アルゴリズムをベースとする．BSS アルゴリズムでは，システムの設計段階で，アプリケーションごとにプロセッサ利用率を割り当てる．実行時には，各アプリケーションのローカルスケジューラが，タスクの相対デッドラインの情報のみを用いて，実行可能タスクの絶対デッドラインの中でもっとも早い絶対デッドラインを計算し，アプリケーションデッドラインとしてグローバルスケジューラに通知する．グローバルスケジューラは，もっともアプリケーションデッドラインの早い，アプリケーションの最高優先度をもつタスクを実行する．

さらに，アプリケーションごとに，実行可能なプロセッサ時間（これをバジェットと呼ぶ）を管理するために，バジェットリストを管理する．アプリケーション $A_i$ のバジェットリストを構成するバジェット要素は，$l_{ij} = (d_{ij}, b_{ij})$ の 2 項組で表す．ここに，$d_{ij}$ は絶対デッドライン，$b_{ij}$ は，$d_{ij}$ までに実行可能な残りバジェットを示す．バジェットリスト内では，バジェット要素が，デッドラインの早い順に並ぶ．BSS では，バジェットリストに対する操作を 3 つ定義している．

**バジェット要素の追加**

アプリケーション $A_i$ のローカルスケジューラが，グローバルスケジューラに対してアプリケーションデッドライン $d_{ij}$ を通知した際に，バジェットリストに $d_{ij}$ に対応するバジェット要素がない場合，新しい要素 $l_{ij}$ を生成して，バジェットリストに追加する．追加処理は，以下の順序で実行する．

(1) 次の条件を満たす，$l_{ij}$ の挿入位置を特定する．

$$\exists l_{i(k-1)}, l_{ik}\ d_{i(k-1)} < d_{ij} < d_{ik}$$

(2) 次の式にしたがって，$d_{ij}$ までに利用可能なバジェットを計算する．

$$b_{ij} = \min\{(d_{ij} - t_c) * U_i, (d_{ij} - d_{i(j-1)}) * U_i + b_{i(k-1)}, b_{ik}\}$$

ここに，$U_i$ は，アプリケーション $A_i$ のプロセッサ利用率，$t_c$ は現在のシステム時刻である．

(3) $l_{ij}$ を，$l_{i(k-1)}$ と $l_{ik}$ の間に挿入する．

**バジェットリストの更新**

以下のイベントが発生したとき，バジェットリストを更新する．

- タスクの実行を完了した
- アプリケーションのバジェットが 0 になった
- アプリケーションデッドラインの早い，別のアプリケーションが実行可能になったために，アプリケーションを切り替える

バジェットリストを更新する場合，アプリケーションデッドラインに対応するバジェット要素 $l_{ij}$ がバジェットのリストの $k$ 番目にあるとすると，以下の処理を実行する．

- $b_{ij} = b_{ij}$ - $e$ (j $\geq$ k)
- $l_{ij}$ を削除する．(j $<$ k $\wedge$ $b_{ij} > b_{ik}$)

ここに，$e$ はタスクを実行するために使用したバジェットである．

**バジェット要素の削除**

バジェットリストを効率的に管理するため，不要となったバジェット要素を，バジェットリストから削除する．ある時刻 $t$ で，デッドラインが $d_{ik}$ に一致するタスクがすでに実行を完了しており，かつ，以下のいずれかの条件を満たすとき，その要素 $l_ik$ を削除する．



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

- $d_{ik} \leq t$
- $b_{ik} > (d_{ik} - t)U_i$

これらの条件は，バジェット割当ての式において，削除できるバジェット要素の情報が利用されないことを示すことで導くことができる[3]．

### 2.3 高優先度タスクによる低優先度タスクのバジェット先使い

BSS アルゴリズム（および，それを改良した PShED アルゴリズム[4]）は，統合プロセッサでのアプリケーションの振る舞いについて，次の２つの性質を保証している．

(1) アプリケーションは，他のアプリケーションの動作の影響を受けることなく，割り当てられたプロセッサ利用率分のバジェットが得られ，アプリケーションデッドラインまでに使い切ることができる．（プロセッサ利用率の保証）

(2) ローカルスケジューリングアルゴリズムとして，EDF を採用するアプリケーションについて，統合前にスケジュール可能なタスクは，統合後もスケジュール可能である．（時間保護の実現）

(2) は，ローカルスケジューリングアルゴリズムに制約を与えており，一般的なリアルタイム OS で採用されている固定優先度ベーススケジューリングには対応していない．

例えば，$\tau_j$ と $\tau_i$ の２つのタスクで構成されるアプリケーションを考える．高い優先度をもつ $\tau_j$ は，起動周期は 5，最悪実行時間は 3，相対デッドラインは起動周期に一致する．$\tau_i$ は，起動周期は 12，最悪実行時間は 4，相対デッドラインは起動周期に一致する．これらのタスクは，固定優先度ベーススケジューリング（例えば RM）でスケジュール可能であり，性能 0.5 の個別プロセッサで，図 1(a) のようにスケジュールされるものとする．このアプリケーションを，性能 1 の統合プロセッサで，50%のプロセッサ利用率を割り当てて実行すると，図 2(a) に示すように，デッドラインをミスしてしまう．これは，ローカルスケジューリングアルゴリズムに固定優先度ベーススケジューリングを採用した場合に，時間保護を実現できていないことを意味している．

個別プロセッサでスケジュール可能であることが検証されたアプリケーションのタスクが，統合プロセッサではデッドラインをミスしてしまう原因を考える．先の例で，低優先度タスク $\tau_i$ が，そのデッドライン 12 までに，高優先度タスク $\tau_j$ に実行を邪魔される時間に着目すると，図 1(b) に示すように，個別プロセッサで邪魔された時間は 8 であったのに対して，統合プロセッサでは，それを越える 4.5（プロセッサの性能比を考慮すると，個別プロセッサにおける時間では 9 に相当する処理量）になっている．つまり，アプリケーションデッドラインより遅いデッドラインをもつ高優先度タスク $\tau_j$ が，アプリケーションデッドラインに一致するデッドラインをもつ低優先度タスク $\tau_i$ に優先して実行されることで，本来は低優先度タスクが使用すべきバジェットを，先使いしてしまったと考えることができる．そこで，アプリケーションデッドラインより遅いデッドラインをもつ高優先度タスクの起動時刻を遅延させることで，低優先度タスクがデッドラインをミスすることを防ぐよう，BSS アルゴリズムを拡張する．

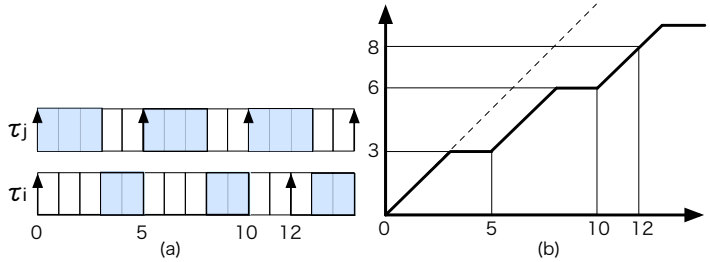


**図 1** 個別プロセッサでデッドラインを満たせるアプリケーション．(a) はスケジュール例，(b)$\tau_i$ が $\tau_j$ に実行を邪魔された時間．

Fig. 1 An application which is schedulable on a dedicated processor. (a):An example of schedule of an application, (b):Interference function of $\tau_j$.

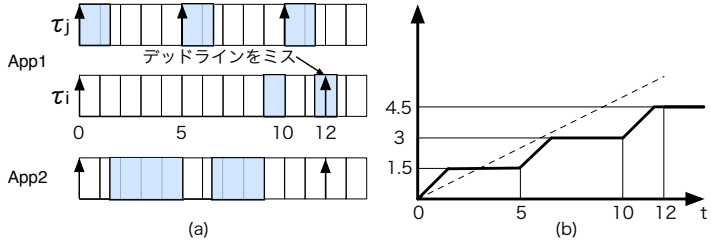


**図 2** BSS でデッドラインミスが発生するアプリケーション．(a) はスケジュール例，(b)$\tau_i$ が $\tau_j$ に実行を邪魔された時間．

Fig. 2 An application with a deadline miss. (a):An example of schedule of an application scheduled by BSS algorithm, (b):Interference function of $\tau_j$.



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

## 3. スケジューリングアルゴリズム

### 3.1 準　　備

BSS アルゴリズムにおいて，ローカルスケジューリングアルゴリズムとして固定優先度ベーススケジューリングを想定して，時間保護を実現できるスケジューリングアルゴリズムについて述べる．統合プロセッサで実行している，あるアプリケーションのデッドラインに一致するタスクを $\tau_i$ で表し，その起動時刻，絶対デッドライン，統合前の最悪実行時間を，それぞれ $R_i$，$D_i$，$C_i$ で表す．

ここで，統合プロセッサでのシステム時刻 $t$ に対して，仮想時刻 $vt(t)$ を導入する．$vt(t)$ は，統合プロセッサにおいて，あるシステム時刻 $t$ までに得られた処理量に応じて決定する仮想的な時間で，統合プロセッサで時刻 $t$ までに得られた処理量を $b$ とすると，$vt(t) = b/U$ で計算する．すなわち，$v(t)$ は，統合プロセッサで得られた場合と同一の処理量を，個別プロセッサで実行した際に得られる時刻を示す．例えば，[0,10] において，U=0.5 のアプリケーションが時刻 5 までに，バジェットを 3 使用したとすると，$vt(5) = b/U = 6$ となる．これは，時刻 5 の時点で，個別プロセッサで時刻 6 までに得られる処理量をすでに使用したことを意味する．

### 3.2 タスク起動時刻の遅延

BSS アルゴリズムでは，2.2 節で述べたように，タスクが起動するとそのタスクの絶対デッドラインを計算し，現在のアプリケーションデッドラインより早い場合には，アプリケーションデッドラインを更新する．さらに，実行中のタスクと優先度を比較して，起動したタスクの方が優先度が高ければ，実行するタスクを切り替える．

提案アルゴリズムでは，アプリケーションデッドラインに一致するタスク $\tau_i$ よりも高い優先度をもち，かつ絶対デッドラインが $D_i$ より遅いデッドラインをもつタスク $\tau_j$ が時刻 $R_j$ で起動したとき，その起動時刻を $R_j = vt(R'_j)$ を満たす $R'_j$ まで遅延する．$R'_j$ は，$R_j$ のみを用いて $R'_j = R_j + (R_j - vt(R_j)) * U$ で計算できる．

### 3.3 動　作　例

図 1 のアプリケーションの提案アルゴリズムによるスケジュール例を図 3 に示す．提案アルゴリズムでは，時刻 10 において $\tau_j$ が起動した場合，タスクの起動時刻を遅延する条件を満たすために，$\tau_j$ の起動時刻を時刻 $11(= 10 + (10 - 8) * 0.5)$ まで遅延させる．この遅延により，$\tau_i$ を実行できるため，$\tau_i$ もデッドラインまでに処理を完了できる．仮想時刻の定義から，$R'_j$ までに使用したバジェットは，個別プロセッサでの時刻 $R_j$ までに得たプロセッサ時間の処理量に一致する．同様に，$R'_j$ の時点で，$D_i$ までに残っているバジェットは，個別プロセッサでの $R_j$ から $D_i$ までのプロセッサ時間に一致する．既存のリアルタイム OS において，このアルゴリズムをどのように実現するかは今後の課題である．

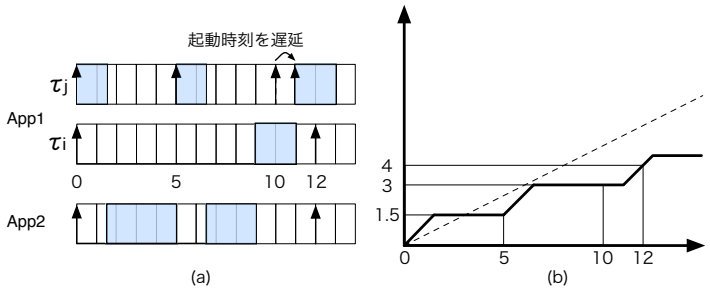


**図 3**　提案アルゴリズムの動作例．(a) はスケジュール例，(b)$\tau_i$ が $\tau_j$ に実行を邪魔された時間．
Fig. 3　Behavior of the proposed scheduling algorithm. (a):An example of schedule of an application scheduled by the proposed algorithm, (b):Interference function of $\tau_j$.

## 4. 証　　明

本章では，個別プロセッサにおいてスケジュール可能なタスク $\tau_i$ が，提案アルゴリズムにより，統合プロセッサでもスケジュール可能であることを示す．アプリケーションのデッドラインに一致するデッドラインをもつタスク $\tau_i$ に着目し，$\tau_i$ の起動時刻から，デッドラインまでの間に，同一アプリケーションに属する高優先度タスクに実行を邪魔される時間が，統合前と統合後で変化しないことを示す．これを示すことができると，$\tau_i$ の起動時刻からデッドラインまでの間に得られる処理量も変化しないため，個別プロセッサにおいてスケジュール可能であれば，統合プロセッサでもスケジュール可能であるといえる．

まず，統合前の個別プロセッサにおいて，$\tau_i$ の実行が同一アプリケーションの別のタスクに邪魔される時間について，次の 2 つの関数を定義する．

- $I_i^a(t_1, t_2, t_3)$
  $[t_1, t_2]$ において，$\tau_i$ より高い優先度をもち，かつ，デッドラインが $t_3$ と同じか早いタスク $\tau_j (\in hp(i) \wedge D_j <= t_3)$ に邪魔される時間を示す．ここに，$hp(i)$ は，タスク $\tau_i$ と同じアプリケーションに属している，$\tau_i$ より高い優先度をもつタスクの集合である．
- $I_i^b(t_1, t_2, t_3)$



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

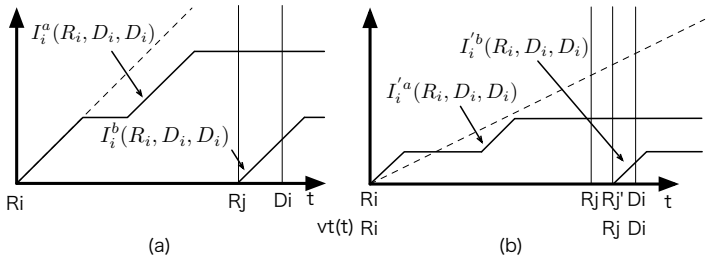


**図 4** 邪魔関数の分割
Fig. 4 Devided interference functions.

タスク $\tau_i$ より高い優先度をもち，かつ，デッドラインが $t_3$ より遅いタスク $\tau_j(\in hp(i) \wedge D_j > t_3)$ に邪魔される時間を示す．

同様に，統合後の統合プロセッサにおいて，$[t_1,t_2]$ 間で，$\tau_i$ の実行が同一アプリケーションの別のタスクに邪魔される時間を，$I_i^{'a}(t_1,t_2,t_3)$ と $I_i^{'b}(t_1,t_2,t_3)$ で表す．これらの定義を用いると，図 1(b) と図 3(b) の邪魔関数を，それぞれ，図 4 の (a) と (b) のように分離できる．

まず，$I_i^a$ と $I_i^b$ の定義から，明らかに次の補題が成り立つ．

**補題 1** 統合前の個別プロセッサにおいて，あるアプリケーションのタスク $\tau_i$ がスケジュール可能であるとき，次の式が成り立つ．

$$I_i^a(R_i,D_i,D_i) + I_i^b(R_i,D_i,D_i) + C_i <= D_i - R_i$$

さらに，統合後において，次の補題が成り立つ．

**補題 2** 区間 $[R_i,D_i]$ において，次の式が成り立つ．

$$I_i^a(R_i,D_i,D_i) * U = I_i^{'a}(R_i,D_i,D_i)$$

**（証明）**簡単のため，すべてのタスクが周期タスクである場合を考える．$\tau_j$ は，$D_j$ までに実行が完了し，かつ $D_j$ は $D_i$ と同じか早いので，$\tau_j$ の起動回数は，$\lfloor \frac{D_i-R_i}{T_j} \rfloor$ で計算できる．$\tau_j$ が一回実行するために使用するバジェット（$\tau_i$ を邪魔する時間）は $C_j$ であるので，合計すると，

$$I_i^a(R_i,D_i,D_i) = \sum_{j \in hp(i)} \lfloor \frac{D_i - R_i}{T_j} \rfloor * C_j$$

となる．$[R_i,D_i]$ でのタスクの起動回数は，統合の前後で変化しないので，統合プロセッサでは，次のようになる．

$$I_i^{'a}(R_i,D_i,D_i) = \sum_{j \in hp(i)} \lfloor \frac{D_i - R_i}{T_j} \rfloor * C_j * U$$

なお，$R_i$ の時点で，実行中のタスク $\tau_k$ が存在する場合には，$\tau_k$ の邪魔時間を $R_i$ の前後に分割して，上記の邪魔時間に加算すれば良い．よって，補題は成立する．

さらに，$Ii^a$ と $Ii^b$ の加算について次の補題が成立する．

**補題 3** $I_i^a(t_1,t_2,t_3)$ は，$t_1 = vt(t_1), t_3 = vt(t_3)$ なる $[t_1,t_3]$ における任意の t について，以下が成り立つ．

$$I_i^a(t_1,t,t_3) = I_i^a(t_1,t,t_3) + I_i^a(t,t_3,t_3)$$

**（証明）** $[t_1,t_3]$ において，デッドラインが $t_3$ と同じか前にある，実行可能なタスクの集合を，$\Phi(t_1,t_3)$ と書く．$\Phi(t_1,t_3)$ に属するすべてのタスク $\tau_j$ について，以下のように分類する．

$$\forall \tau_j \in \Phi(t_1,t_3), \begin{cases} \tau_j \in \Phi(t_1,t) & (if\ D_j \le t) \\ \tau_j \in \Phi(t,t_3) & (if\ D_j > t) \end{cases}$$

明らかに，$\Phi(t_1,t_3) = \Phi(t_1,t) + \Phi(t,t_3)$ が成り立つ．よって，実行可能タスクによる邪魔時間 $I_i^a(t_1,t_2,t_3)$ についても，

$$I_i^a(t_1,t,t_3) = I_i^a(t_1,t,t_3) + I_i^a(t,t_3,t_3)$$

が成り立つことから，補題が示される．

$Ii^b()$ について次の補題が成立する．

**補題 4** $t_1 = vt(t_1), t_2 = vt(t_2)$ が成立する区間 $[t_1,t_2]$ で，任意の時刻 $t$ ついて，次の式が成り立つ．

$$I_i^b(t_1,vt(t),t_2) * U = I_i^{'b}(t_1,t,t_2) \quad (1)$$

$$I_i^b(vt(t),t_2,t_2) * U = I_i^{'b}(t,t_2,t_2) \quad (2)$$

さらに，

$$I_i^b(t_1,t_2,t_2) * U = I_i^{'b}(t_1,t_2,t_2)$$

が成り立つ．

**（証明）**提案アルゴリズムでは，統合プロセッサでの時刻 $t$ で残っているバジェットは，個別プロセッサでの時刻 $vt(t)$ での残りプロセッサ時間の処理量に一致する．よって，式 (2) が成立することは明らかである．また，

$$I_i^b(t_1,vt(t),t_2) * U = I_i(t_1,t_2) * U - I_i^a(t_1,t_2) * U - I_i^b(vt(t),t_2,t_2)$$

$$I_i^{'b}(t_1,t,t_2) = I_i'(t_1,t_2) - I_i^{'a}(t_1,t_2) - I_i^b(t,t_2,t_2)$$

が成り立つ．ここに，$I_i(t_1,t_2)$ と $I_i'(t_1,t_2)$ は，$[t_1,t_2]$ での合計邪魔時間である．$t_1 = vt(t_1)$,



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

$t_2 = vt(t_2)$，さらに補題２，式 (2) より，上式の右辺が等しいことから，

$$I_i^b(t_1, vt(t), t_2) * U = I_i^{'b}(t_1, t, t_2)$$

となる．よって，任意の $[t_1, t]$ においては，

$$I_i^b(t_1, vt(t), t_2) * U = I_i^{'b}(t_1, t, t_2)$$

が成立する．さらに，$[t_1, t_2]$ でアプリケーションに割り当てられるバジェットが $(t_2 - t_1) * U$ であることと，補題２より

$$I_i^b(t_1, t_2, t_2) * U = I_i^{'b}(t_1, t_2, t_2)$$

が成立する．

最後に，以上の補題を用いて，次の定理が成り立つことを示す．

**定理１**　統合前に性能 U の個別プロセッサで，固定優先度スケジューリングによりスケジュール可能なアプリケーションのタスク $\tau_i$ は，統合プロセッサで提案アルゴリズムによりデッドランを満たす．

**（証明）**統合プロセッサでアプリケーションがプロセッサを与えられる状況により，証明方法が異なる．ここでは，$[R_i, D_i]$ において，アプリケーションがスケジュールされる場合を２つに分けて，それぞれの状況において，定理が成り立つことを示す．

場合１：t ≤ vt(t)

まず，統合プロセッサでの処理量が，常に仮想時刻と同じ時刻か早い時刻で得られる場合を考える．この状況は，$[R_i, D_i]$ において，アプリケーションが他のアプリケーションに優先してスケジュールされる場合であり，$\tau_i$ と，$\tau_i$ より高い優先度をもつ実行可能なタスク $\tau_j$ に対して，個別プロセッサで実行するよりも早い時刻でプロセッサが割り当てられる．ここで，$[R_i, D_i]$ で，$\tau_i$ の実行が完了するまでに，統合プロセッサで保証されるプロセッサ時間は $(D_i - R_i) * U$ である．

いま，$\tau_i$ は個別プロセッサでデッドラインを満たすことから，$\tau_i$ の実行完了時刻 $F_i$ では，$[R_i, F_i]$ に実行可能になった高優先度タスク $\tau_j (\in hp(i))$ はすべて実行が完了している．一方，統合プロセッサでも，$vt(t') = F_i$ となる時刻 $t'$ で，個別プロセッサにおいて $\tau_i$ の実行を完了するために必要なプロセッサ時間をすでに得ていることになる．このとき，統合プロセッサにおいて，$[R_i, t']$ で実行可能な高優先度タスク集合 $\tau_j \in hp(i)$ は，個別プロセッサで $[R_i, F_i]$ で実行可能になった高優先度タスク集合 $\tau_j (\in hp(i))$ に一致するか，より少ないはずである．よって，統合プロセッサでは，$t'$ までに $\tau_i$ の実行は完了することから，個別プロセッサでデッドラインを満たす $\tau_i$ は．統合プロセッサでもデッドラインを満たせる．

場合 2: t > vt(t)

統合プロセッサでのシステム時刻が，仮想時刻より遅い時刻になる場合，提案アルゴリズムでは，$D_i$ より遅いデッドラインをもつタスク $\tau_j$ の起動時刻 $R_j$ を $R'_j (= vt(R_j))$ まで遅延する．このとき，定理１を背理法を用いて証明する．すなわち，個別プロセッサでスケジュール可能である $\tau_i$ が，統合プロセッサでデッドラインをミスすると仮定し，矛盾を導く．まず，個別プロセッサで $\tau_i$ はスケジュール可能であることから，補題１より，

$$I_i^a(R_i, D_i, D_i) + I_i^b(R_i, D_i, D_i) + C_i <= D_i - R_i$$

が成り立つ．さらに，補題３と補題４から，$\tau_j$ の起動時刻 $R_j$ の前後に分割して次のように変換できる．

$$I_i^a(R_i, R_j, R_j) + I_i^a(R_j, D_i, D_i) + I_i^b(R_i, R_j, D_i) + I_i^b(R_j, D_i, D_i) + C_i <= D_i - R_i$$

一方，統合プロセッサでは，$\tau_i$ がデッドラインをミスすると仮定しているので，

$$I_i^{'a}(R_i, D_i, D_i) + I_i^{'b}(R_i, D_i, D_i) + C_i * U > (D_i - R_i) * U$$

となる．$I_i^{'a}$ は $R_j$ と $R'_j$ で，$I_i^{'b}$ は，$R'_j$ でそれぞれ分割すると，

$$I_i^{'a}(R_i, R_j, R_j) + I_i^{'a}(R_j, R'_j, R'_j) + I_i^{'a}(R'_j, D_i, D_i)$$
$$+ I_i^{'b}(R_i, R'_j, D_i) + I_i^{'b}(R'_j, D_i, D_i) + C_i * U > (D_i - R_i) * U$$

となる．補題３より，$R_j = vt(R'_j)$ では，処理量が一致することから，

$$I_i^a(R_i, R_j, R_j) * U + I_i^b(R_i, R_j, D_i) * U$$
$$= I_i^{'a}(R_i, R_j, R_j) + I_i^{'a}(R_j, R'_j, R'_j) + I_i^{'b}(R_i, R'_j, D_i)$$
$$I_i^a(R_j, R'_j, R'_j) * U + I_i^a(R'_j, D_i, D_i) * U + I_i^b(R_j, D_i, D_i) * U$$
$$= I_i^{'a}(R'_j, D_i, D_i) + I_i^{'b}(R'_j, D_i, D_i)$$

がそれぞれ成り立つ．これらを整理すると，

$$I_i^a(R_i, R_j, R_j) + I_i^b(R_i, R_j, D_i) + I_i^a(R_j, R'_j, R'_j) +$$
$$I_i^a(R'_j, D_i, D_i) + I_i^b(R_j, D_i, D_i) + C_i > (D_i - R_i)$$

となる．統合前の条件式と，統合後の条件式を整理すると，

$$I_i^a(R_j, R'_j, R'_j) + I_i^a(R'_j, D_i, D_i) > I_i^a(R_j, D_i, D_i)$$

となる．これは明らかに補題３に矛盾することから，定理１が成立する．

## 5. 結　　論

本論文では，タスクのデッドラインの情報のみを用いる BSS アルゴリズムを拡張することで，時間保護を実現できる階層型スケジューリングを提案した．具体的には，まず，デッドラインの長い高優先度タスクが，デッドラインの短い低優先度タスクの実行時間を奪うことが，デッドラインミスの原因であることを指摘した．そして，デッドラインの遅い高優先



ⓒ2010 Information Processing Society of Japan

Vol.2010-ARC-187 No.25
Vol.2010-EMB-15 No.25
2010/1/29

度タスクの起動時刻を遅延させることで，低優先度タスクがデッドラインをミスすることを防ぐようアルゴリズムを拡張した．最後に，提案アルゴリズムにより，統合前にスケジュール可能なタスクは，統合後にもスケジュール可能であることを証明した．今後は，提案アルゴリズムを既存のリアルタイム OS へ実装し，実装オーバヘッドを測定する予定である．

## 参 考 文 献

1) Z.Deng and J.W.-S.Liu and L.Zhang and S.Mouna and A.Frei, *An Open Environment for Real-Time Applications*, Real-Time Systems Journal, 16, pp.155-185, (1999).
2) 松原豊，本田晋也，冨山宏之，高田広章：時間保護のためのリアルタイムスケジューリングアルゴリズム，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.SIG 8(ACS18), pp.192-202, (2007).
3) G.Lipari and K.Baruah, *Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems*, In Proceedings of IEEE Real-Time Technology and Applications Symposium, (2000).
4) G. Lipari and J. Carpenter and S. Baruah, *A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments*, In Proceedings of the Real-time System Symposium, IEEE Computer Society Press, (2000).
5) G.Lipari, *Resource Reservation in Real-Time Systems*, Ph.D Thesis, Scuola Superiore S.Anna, Pisa, Italy, (2000).
6) I.Shin and I.Lee, *Compositional Real-time Scheduling Framework*, In Proceedings of IEEE Real-time Systems Symposium, pp.57-67, (2004).



ⓒ2010 Information Processing Society of Japan