社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8（30）
2008 / 3 / 28

タスク優先度を考慮した時間保護スケジューリングアルゴリズム

松原　豊†　本田　晋也†　冨山　宏之†　高田　広章†

† 名古屋大学大学院情報科学研究科　〒 464-0804 愛知県名古屋市千種区不老町
E-mail: †{yutaka,honda,tomiyama,hiro}@ertl.jp

あらまし　本論文では，複数の組込みリアルタイムアプリケーションが動作するシングルプロセッサシステム向けに，アプリケーション毎のプロセッサ時間を保護するための効率的な時間保護スケジューリングアルゴリズムを提案する．これまでに提案された時間保護アルゴリズムは，タスクの起動やデッドラインなどのタイミングでアプリケーションにプロセッサ時間を割り当てるため，アプリケーションの切替え回数が多くなる問題があった．提案アルゴリズムでは，プロセッサ時間と，そのプロセッサ時間を使用するタスクの優先度を関連付けて管理することで，不要な割当てをなくす．これにより，時間保護を実現しつつ，アプリケーション切り替え回数を削減できることを簡単なアプリケーションを用いて示す．

キーワード　時間保護，リアルタイムスケジューリング，リアルタイム OS

An Efficient Real-Time Scheduling Algorithm for Temporal Protection with Task's Priority.

Yutaka MATSUBARA†, Shinya HONDA†, Hiroyuki TOMIYAMA†, and Hiroaki TAKADA†

† Graduate School of Information Science, Nagoya University　Furo-cho, Chikusa-ku, Nagoya-shi, Aichi, 464-0804 Japan
E-mail: †{yutaka,honda,tomiyama,hiro}@ertl.jp

Abstract　In this paper, we propose an efficient scheduling algorithm for integration of real-time applications in a single processor on the assumption that each application could satisfy its timing constraints in a dedicated processor. In some conventional algorithms, the overhead of switching applications tends to increase because the global scheduler, which decide which application to be executed, allocates processing time to each application at every application events which consist of task's release time and deadlines. On the other hands, in the proposed algorithm, the global scheduler allocates processing time to each application at only when necessary by using task's priority. we show that the grain size of each allocated processing time could be increase and the number of application switching times decreases by using a example of simple applications.

Key words　Temporal Protection, Real-Time Scheduling, Real-Time OS

1. はじめに

近年，ハードリアルタイム性を要求される代表的なシステムの一つである自動車制御システムの開発においては，その高性能化・複雑化により，ECU （Electronic Control Unit）と呼ばれる電子制御ユニットの数が増加し，コストの増加，設置スペースの不足が大きな問題となっている．システムに搭載される ECU 数を削減する手法としては，複数の ECU でそれぞれ動作しているアプリケーションを単一の ECU 上で動作させることが有効な手法の一つと考えられる．

複数のリアルアプリケーションを単一のプロセッサに統合する場合，あるアプリケーションが想定した以上のプロセッサ時間（これをバジェットと呼ぶ）を使ったために，他のアプリケーションが時間制約を満たせなくなることを防ぐ必要がある．また，統合前に時間制約を満たして動作するアプリケーションが，統合することで時間的な振る舞いが変化し，その結果時間制約を満たせなくなる場合があると，統合に際してアプリケーションの再検証（場合によっては再設計も）が必要となり，統合を進めることが困難になる．そこで我々は，リアルタイムアプリケーションの統合を容易に進める環境を構築するため，リアルタイム OS のスケジューラに必要な機能として，「統合前に時間制約を満たすリアルタイムアプリケーションは，統合後にも時

間制約を満たすことを保証する機能」を時間保護と定義し，これを実現するスケジューリングアルゴリズムと，既存のリアルタイム OS を拡張したスケジューリングフレームワークを提案した [1], [2].

これまでの時間保護アルゴリズムでは，アプリケーションに対して，所属するタスクのイベント（起動時刻とデッドライン）のすべてのタイミングで，次のイベントまでに利用できるバジェットを割り当てる．そのため，発生時刻の近いイベントがあると，アプリケーションの切替え回数が増加する問題があった．たとえば，高い優先度をもつタスクの実行中に，そのタスクより優先度の低いタスクが起動する場合，統合前の個別プロセッサではタスクの切替えは発生しない．このような状況では，統合プロセッサにおいても低いタスクの実行が完了するまでアプリケーションの実行が継続されるのが理想的であるが，従来のアルゴリズムを適用すると，タスクの優先度を考慮せずに無条件にバジェットを割当てるため，高優先度タスクの起動時刻に低優先度タスクの起動までのバジェット割当てる．その結果，低優先度タスクが起動する前にバジェットがなくなり，無駄なアプリケーション切り替えが発生してしまう．

本論文では，タスク優先度を考慮することで，アプリケーション切り替え回数を削減する効率的な時間保護スケジューリングアルゴリズムを提案する．提案アルゴリズムでは，バジェットと，そのバジェットを使用するタスクの優先度を関連付けて管理することで，割当てるバジェットの粒度を大きくする．たとえば，個別プロセッサでタスク切り替えが発生しないタイミングにおいてはバジェットを分割しない．その結果，アプリケーションの切り替え回数を削減できる．

以下，本論文の構成を述べる．2. 章では，リアルタイムアプリケーションの状況を整理し，時間保護の必要を述べる．3. 章では，提案アルゴリズムについて詳細に述べ，4. 章で，提案アルゴリズムによるアプリケーションの具体的な動作例を示す．5. 章では，関連研究について述べ，提案手法の位置づけを明確に述べる．最後に，6. 章で本論文の結論と今後の課題を述べる．

2. リアルタイムアプリケーション統合

2.1 状況設定

本論文では，図 1 に示すように，複数の個別プロセッサ上で時間制約を満たして動作するアプリケーションを，単一の統合プロセッサ上に統合する状況を想定する．ここで，個別プロセッサとは，1 つのアプリケーションのみを動作させるための統合前のプロセッサのことを，統合プロセッサとは，複数のアプリケーションを動作させるための統合後のプロセッサのことをいう．通常，統合プロセッサは，個別プロセッサに比べて高い処理性能をもつ．各アプリケーションには，個別プロセッサの性能に応じてプロセッサ利用率（シェアと呼ぶ）を設定する．なお，プロセッサ性能と処理量の関係は単純化し，単位時間当たりの処理量はプロセッサの性能に比例するものとする．すなわち，性能 1 のプロセッサで 2 単位時間で実行を完了するタスクは，性能 2 のプロセッサでは，1 単位時間で実行を完了できるものとする．

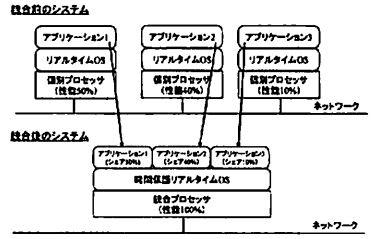

図 1 リアルタイムアプリケーション統合

Fig. 1 Integration of real-time applications on a single processor.

2.2 プロセッサ時間の保護機能

組込みシステムのアプリケーションは，要求される時間要件により，所属するタスクが厳密なデッドラインをもつリアルタイムアプリケーションと，厳密なデッドラインを持たない非リアルタイムアプリケーションに分類できる．非リアルタイムアプリケーションに対しては，アプリケーションのプロセッサ利用率を保証できれば十分である場合が多い．統合プロセッサにおけるアプリケーションのプロセッサ利用率を保護することで，統合前に得られる処理量が，ある時間内のどこかで得られることが保証される．この場合，統合プロセッサでのアプリケーションの実行順序は他のアプリケーションに影響され，バジェットが得られる時刻は保証されない．そのため，リアルタイムアプリケーションに対しては不十分である．

リアルタイムアプリケーションを統合する場合，アプリケーションのプロセッサ利用率の保証に加えて，そのアプリケーションに所属するタスクが時間制約を満たすことを保証する時間保護が適する．時間保護により，個別プロセッサにおけるアプリケーションの動作検証の結果は，統合後の統合プロセッサでも有効となる．本論文では，リアルタイムアプリケーションの統合を前提として，時間保護を実現するスケジューリングアルゴリズムを対象とする．

2.3 QoS 制御タスク

自動車制御システムなど実際のリアルタイムシステムでは，システムの負荷に応じて処理内容を変化させている場合など，統合プロセッサで他のアプリケーションと統合して実行した場合に，個別プロセッサで実行したときと比べて，要求する処理量が変化するタスクが存在する．このようなタスクを QoS 制御タスクと呼ぶ．たとえば，QoS 制御タスクは，実行中にデッドラインになった場合にはただちに実行を終了するが，他に実行可能タスクが存在しない場合にはさらにいくらか実行を継続する．プロセッサがアイドル状態にあるときに実行するような故障診断タスクやバックグラウンドタスクなども QoS 制御タスクに該当する．

QoS 制御タスクが含まれるアプリケーションに対しては，従来の時間保護アルゴリズムを適用することで時間保護を実現できる．たとえば，個別プロセッサにおいて図 2 のようなに動作する 2 つのアプリケーションを考える．図中の上方向矢印，下向き矢印，下線は，それぞれタスクのリリース時刻，デッドライン，実行可能であることを示す．一つ目のアプリケーション

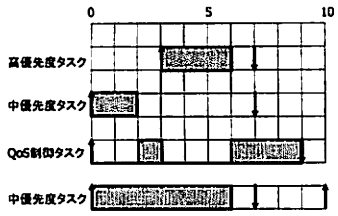

図 2　サンプルアプリケーション
Fig. 2　sample applications.

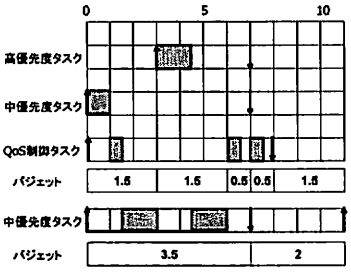

図 3　従来の時間保護アルゴリズムによるスケジューリング
Fig. 3　An example of schedule produced by the conventional algorithm.

(アプリケーション A とする) は，高優先度タスク，中優先度タスク，QoS 制御タスクの 3 つのタスクで構成され，固定優先度スケジューリングアルゴリズムでスケジュールする．二つ目のアプリケーション (アプリケーション B とする) は，1 つの中優先度タスクで構成され，スケジューリングアルゴリズムはない．どちらも，個別プロセッサでは時間制約を満たして実行できる．

この 2 つのアプリケーションを，個別プロセッサの 2 倍の性能をもつ統合プロセッサに統合し，従来の時間保護アルゴリズムを適用する．各アプリケーションに，それぞれプロセッサ利用率 50%を割当てると図 3 のようにスケジュールされ，各アプリケーションは時間制約を満たして実行できる．時間保護アルゴリズムは，各タスクの起動時刻とデッドラインのすべてのタイミングで，次のイベントまでの時間とシェアの積をバジェットとして割り当てるため，アプリケーションの切替え回数が多くなる問題がある．

QoS 制御されたタスクを含まないアプリケーションに対しては，PShED アルゴリズムで時間保護を実現できることが分かっている [3]〜[5]．PShED アルゴリズムでは，タスクのデッドラインのみを用いて，各タスクがデッドラインを満たすようにスケジュールできる．さらに，タスクごとに利用可能なバジェットをリストとして管理することで，従来の時間保護アルゴリズムに比較してアプリケーションの切り替え回数が少ないという特徴をもつ．しかしながら，アプリケーション内に QoS 制御タスクが存在すると場合には，図 4 に示すように，統合後に一部のタスクが時間制約を満たせない場合がある．

このように，QoS 制御タスクにより，将来起動するタスクのプロセッサ時間が先使いされ，高優先度タスクがデッドライン

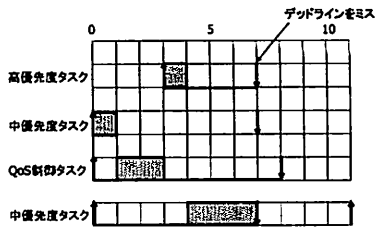

図 4　PShED アルゴリズムによるスケジューリング
Fig. 4　An example of schedule produced by the PShED algorithm.

をミスしてしまう状況を防ぐためには，タスクの起動時刻をスケジューラが把握し，将来起動する高優先度タスクが時間制約を満たすために必要なプロセッサ時間を残しておく必要がある．本論文では，アプリケーションの切り替え回数が少なくて済む PShED アルゴリズムをベースに，QoS 制御されたタスクが含まれるアプリケーションに対しても適用可能なスケジューリングアルゴリズムを提案する．

2.4　バジェット先使いの条件

PShED アルゴリズムにおいて，QoS 制御タスクが優先度の高いタスクのバジェットを先使いする状況を考察し，その発生条件を以下の 2 つに整理する．

- 条件 1：実行するタスクが QoS 制御タスクであること．
- 条件 2：実行するタスクの優先度が，個別プロセッサにおいてバジェットを使用するタスクの優先度より低いこと．

タスク切り替え時を含め，タスクの実行を開始する際にこの 2 つの条件が成立する場合，バジェットの先使いが発生する可能性がある．逆に言えば，この 2 つの条件のうち，どちらか一方でも成立しない場合は，バジェットの先使いが発生しないことになる．そこで，PShED アルゴリズムにおいて，条件 1 と条件 2 が両方成立する状況では，実行するタスクより優先度の高いタスクの起動時刻をバジェットの計算で考慮するよう拡張する．さらに，バジェットを使用できるタスクの優先度とバジェットとを関連付けて管理する．条件 1 もしくは条件 2 のどちらか一方のみが成立する状況においては，PShED アルゴリズムのバジェット計算方法を適用する．

3.　スケジューリングアルゴリズム

3.1　階層型スケジューラ

提案アルゴリズムのスケジューラ構成を図 5 に示す．このスケジューラは，ローカルスケジューラとグローバルスケジューラを 2 階層に配置した階層型スケジューラである．提案アルゴリズムでは，アプリケーション A_i ごとに専用のローカルスケジューラ S_i を割り当てる．システム設計者は，各アプリケーションにシェア U_i を割当てる．ローカルスケジューラは，シェアに超えない範囲のプロセッサ時間を使って，アプリケーション内のタスクをスケジュールする．アプリケーションのシェアの合計は 1 を超えない．

アプリケーション A_i のタスクが実行可能になると，ローカルスケジューラ S_i は，そのスケジューリングポリシーに従って

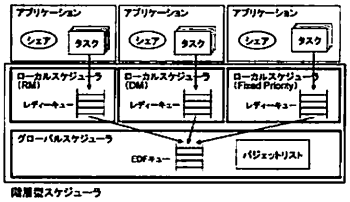

図 5 スケジューラ構成

Fig. 5 Construction of the hierarchical scheduler.

最高優先度タスクを決定し，グローバルスケジューラの EDF キューに挿入する．タスクのスケジューリングポリシーは，個別プロセッサにおけるスケジューリングポリシーと同じで，静的に優先度割付けのスケジューリングアルゴリズムを想定する．さらに，アプリケーションのイベント e を計算する．実行可能な（すなわち，実行可能なタスクをもつ）アプリケーションの最高優先度タスクは，アプリケーションのイベントの早い順番でグローバルスケジューラの EDF キューに挿入される．

グローバルスケジューラは，どのローカルスケジューラの最高優先度タスクを実行するのかを決定するスケジューラで，最も早いイベントをもつアプリケーションを選択し，最大でそのアプリケーションのバジェット分だけ実行する．アプリケーションのイベントとは，実行するタスクのイベントと，すべてのデッドラインの中で最も早い時刻に一致する．タスクのイベントとは，そのタスクより優先度の高いタスクの起動時刻と，そのタスク自身のデッドラインの早い方の時刻を示す．なお，グローバルスケジューラの EDF キューに挿入されるタスクは，ローカルスケジューリングポリシーに基づいて決定されるので，そのタスクのイベントアプリケーションイベントに一致するとは限らないことに注意されたい．

3.2 バジェットリスト

グローバルスケジューラは，アプリケーションのバジェットを計算するために，バジェットリストと呼ぶデータ構造を管理する．アプリケーション A_i のバジェットリスト L_i は，以下の形式の要素を管理する．

$$l = (e, P, B)$$

e はタスクのイベント，P は，e がタスクの起動時刻の場合は，時刻 e で起動するタスクの優先度，デッドラインの場合はアプリケーション内のタスクの最高優先度（ここでは P_{max} とする）である．B は時刻 e までの間に利用可能なバジェットの残量を示す．グローバルスケジューラは，アプリケーションが，バジェットリストの各要素に書かれている残りバジェットを超えて実行されないようにリストを更新する．リスト L_i はイベント e の昇順で整列する．タスクを実行する際は，バジェットリストからそのタスクのイベントに対応するバジェット要素から残りバジェットを取得する．残りバジェットは，アプリケーションを連続で実行できる最長時間となる．バジェットリストを管理する操作としては，新しいバジェット追加，リスト更新，バジェット要素削除の 3 つの操作がある．

3.3 バジェット要素の追加

実行するタスクのイベントに対応するバジェット要素が，バジェットリストにない場合，新しいバジェット要素を生成してバジェットリストに追加する．タスクイベントを e_j とすると，バジェットリストを探索して以下の条件を満たす挿入位置（ここでは k 番目とし，$l_i(k)$ と表す）を決定する．

$$\exists l_i(k-1), l_i(k) \quad e_{k-1} < e_j \leqq e_k$$

次に，バジェット要素 $l_i(k)$ を計算する．バジェット先使いの条件を満たさない場合は，以下の式に基づいてバジェット要素の三項組み (e, P, B) を計算する．

$$\begin{aligned} e &= e_j \\ P &= P_{max} \\ B &= \min\{(e_j - t)U_i, (e_j - e_{k-1})U^i + B_{k-1}, B_k\} \end{aligned}$$

ここに，t はシステム時刻，U_i はアプリケーション A_i のシェアである．一方，バジェット先使いの 2 条件を満たす場合には，以下の式にしたがって計算する．

$$\begin{aligned} e &= e_j \\ P &= P_j \\ B &= \max\{(e_j - a_k)U_i - ((e_k - a_k)U^A - B_k, 0\} \\ &= \max\{B_k - (e_k - e_j)U_i, 0\} \end{aligned}$$

ここに，P_j は e_j で起動するタスクの優先度である．a_k はバジェット要素 $l_i(k)$ がリストに追加された時刻であるが，式展開によりバジェットの計算では不要となる．この新しく計算した 3 項組 $l = (e, P, B)$ をバジェットリストの k 番目に挿入する．

3.4 バジェットリストの更新

実行中タスクが完了すると，そのタスクが所属するアプリケーションのバジェットリストを更新する．リストを更新するタイミングは，以下のいずれかによる．

- 実行中タスクの実行が完了した．
- バジェットがすべて消費された．
- より早いイベントをもつアプリケーションにプリエンプトされた．

これらのとき，アプリケーションイベントに一致する要素をリストから探索する．その要素が k 番目の要素だとすると，以下のようにリストのバジェット要素を更新する．

$$\begin{aligned} &\forall l_j \quad j \geqq k \quad B_j = B_j - exc \\ &\forall l_j \quad j < k \wedge B_j > B_k \rightarrow l_j \text{を削除する．} \end{aligned}$$

ここに，exc はスケジュールされてからの実行時間を示す．さらに，$l_i(k)$ に対応するタスクの実行が完了した場合は，$l_i(k)$ の優先度 P_k をアプリケーション内の最低優先度（ここでは，P_{min} とする）に設定する．これは，以降のバジェット計算において，e_k を考慮する必要がないことを示す．

3.5 バジェットの削除

次に，不要となったバジェット要素を削除するルールを説明

する．時刻 t で，イベントが t に一致するバジェット要素について，対応するタスクの実行が完了しており，かつ，以下のいずれかの条件を満たすとき，その要素 $l_i(k)$ は削除できる．

- $e_k \leqq t_i$
- $B_k > (e_k - t)U_i$

これらの条件はバジェット割当ての式から導くことができる．上記のいずか条件を満たすバジェット要素は，以降のバジェットの計算においては加味されることはないため，削除しても良い．たとえば，要素 $l_i(j)$ が $l_i(k)$ の後に挿入されていると仮定すると，

$$e_j U_i = (e_j - t)U_i < B_k + (e_j - e_k)U_i$$

となり，$B_k + (e_j - e_k)U_i$ 部分は最小値とはならないことから，バジェット計算に影響しないことになる．また，要素 $l_i(j)$ が $l_i(k)$ の前に挿入されていると仮定すると，

$$e_j U_i = (e_j - t)U_i < (e_k - t)U_i < B_k$$

となり，B_k は最小値とはならない．よって，先と同様に $l_i(k)$ は新しいバジェット要素のバジェット計算に影響しないので，その要素は安全に削除できる．

4. 動 作 例

2. 章で取り上げた 2 つのアプリケーションに対して，提案アルゴリズムによるスケジュールの例を図 6 に示す．アプリケーションの動作条件は，2. 章と同様に，各アプリケーションのシェアは 50%，アプリケーション A のスケジューリングポリシーは固定優先度スケジューリングである．

ここでは，アプリケーション A にのみ着目して動作を説明する．時刻 $t = 0$ で中優先度タスクが起動する．中優先度タスクは，QoS 制御タスクではないため，タスクイベントは自身のデッドラインである時刻 7 となる．同時刻には，低優先度タスクも起動し，タスクイベントは，自身より優先度の高いタスクの起動時刻（すなわち，高優先度タスクの起動時刻である時刻 3）と，自身のデッドラインの早い時刻である時刻 3 となる．QoS 制御タスクのイベントは，中優先度タスクよりも早いが，実行するタスクは中優先度タスクであるので，この時点でアプリケーション A のアプリケーションイベントは時刻 7 となる．グローバルスケジューラは，この二つの情報とアプリケーションのシェアから，以下のように新しいバジェットを計算する．

$$\begin{aligned} e &= 7 \\ P &= P_{max} \\ B &= (e_j - t)U_i = 7 * 0.5 = 3.5 \end{aligned}$$

バジェット要素 $l = (7, P_{max}, 3.5)$ を計算し，バジェットリストに挿入する．この時点でバジェットリストは以下のようになる．

$$L^A = \{(7, P_{max}, 3.5)\} \qquad (1)$$

時刻 0 では，アプリケーション A のイベントは，アプリケーション B のイベントである時刻 7 と同じなので，アルゴリズム

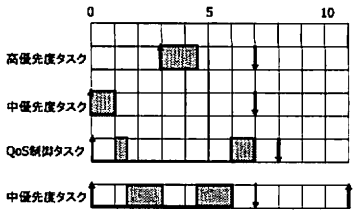

図 6 提案アルゴリズムによるスケジューリング

Fig. 6 An example of schedule produced by the proposed algorithm.

上はどちらを先に実行しても問題ない．ここでは，グローバルスケジューラがアプリケーション A の中優先度タスクを実行することにする．

時刻 $t = 1$ で中優先度タスクの実行が完了すると，グローバルスケジューラはアプリケーション A のバジェットリストに対してバジェットの消費量を反映し，タスクの実行完了による優先度の変更を実施する．その結果，バジェットリストは，以下のようになる．

$$L^A = \{(7, P_{min}, 2.5)\} \qquad (2)$$

さらに，実行するタスクが QoS 制御タスクに切り替わるため，アプリケーションイベントも時刻 3 になる．バジェットリストには，時刻 3 に対応するタスクがないため，以下のように新しいバジェット要素を計算しリストに追加する．

$$\begin{aligned} e &= 3 \\ P &= P_H \\ B &= \max\{B_k - (e_k - e_j)U_i, 0\} \\ &= 2.5 - (7 - 3) * 0.5 \\ &= 0.5 \end{aligned}$$

この結果，この時点でのバジェットリストは以下のようになる．

$$L^A = \{(3, P_H, 0.5), (7, P_{min}, 2.5)\}$$

時刻 $t = 1.5$ になると，アプリケーション A のバジェットが 0 になり，アプリケーション A は実行を継続できない．グローバルスケジューラは，アプリケーション B に実行を切り替える．この時点で，アプリケーション A のバジェットリストは以下のようになる．

$$L^A = \{(3, P_H, 0), (7, P_{min}, 2)\}$$

時刻 $t = 3$ になると，まず，システム時刻とイベント時刻が一致するバジェット要素をリストから削除する．次に，アプリケーション A の高優先度タスクが起動する．高優先度タスクのイベントは，自身のデッドラインである時刻 6 で，追加するバジェット要素は $(6, P_{max}, 1.5)$ である．これらをバジェットリストに反映すると以下のようになる．

$$L^A = \{(6, P_{max}, 1.5), (7, P_{min}, 2)\}$$

この時点で，アプリケーション A のイベントは時刻 6，アプリ

ケーションBのイベントは時刻7であるので，グローバルスケジューラはアプリケーションAの高優先度タスクを実行する．

時刻 $t = 4.5$ で，高優先度タスクの実行が完了すると，バジェットリストは以下のように更新される．

$$L^A = \{(6, P_{min}, 0), (7, P_{min}, 0.5)\}$$

アプリケーションAの最高優先度タスクはQoS制御タスクとなる．この時点で，QoS制御タスクのイベントは，自身のデッドラインである時刻8となり，新しいバジェット要素を追加する．

$$L^A = \{(6, P_{min}, 0), (7, P_{min}, 0.5), (8, P_{max}, 1)\}$$

なお，アプリケーションイベントも時刻8となり，この時点でアプリケーションイベントが時刻7であるアプリケーションBが実行される．

時刻 $t = 6$ になると，まず，システム時刻とイベント時刻が一致するバジェット要素をリストから削除する．アプリケーションBの中優先度タスクの実行が完了するので，次にアプリケーションAのQoS制御タスクの実行を開始する．この時点でのアプリケーションイベントは時刻8なので，時刻8までQoS制御タスクを残りバジェット1だけ実行できる．この時点でのバジェットリストは以下のようになる．

$$L^A = \{(7, P_{min}, 0.5), (8, P_{max}, 1)\}$$

時刻 $t = 7$ になると，アプリケーションAのバジェットが0になり，最終的にバジェットリストは以下のようになる．

$$L^A = \{(8, P_{min}, 0)\}$$

以上より，PShEDアルゴリズムではデッドラインミスが発生したアプリケーションを，提案アルゴリズムを適用することで，時間制約を満たしてスケジュール可能となる．さらに，図3に示した，従来の時間保護アルゴリズムによるスケジュール結果と比較すると，時刻6から時刻8において，従来アルゴリズムではバジェット1を0.5ずつ二回に分けて割当てるのに対して，提案アルゴリズムでは一回でバジェット1を割当てる．提案アルゴリズムを適用することで，バジェットの粒度が大きくなり，アプリケーションの切り替え回数が削減できることが分かる．

5. 関連研究

Z. Dengらは，すべてのタスクの起動時刻とWCETが既知であることを適用条件として，時間保護を実現できるOpen Systemを提案した[7]．我々は，Open Systemをベースに WCETをデッドラインに置き換えることで，より適用条件を緩めた時間保護アルゴリズムを提案した[1]．G. Lipariらは，タスクのデッドラインのみを利用して時間保護を実現できるPShEDアルゴリズムを提案した[5]．適用条件がタスクのデッドラインのみであるため，多くのアプリケーションに適用できると考えるが，QoS制御タスクが存在する場合，それより高い優先度をもつタスクが時間制約を満たせなくなる場合がある．また，G. Lipariらは，タスクの起動周期とWCETを利用して，アプリケーション内のすべてのタスクが時間制約を満たす(P,Q)の組を求める手法も提案している[6]．Pはアプリケーションの実行周期，Qは周期毎に必要なプロセッサ時間である．この手法は，QoS制御タスクが存在する場合にも時間保護を実現でき，実行時にはすべてのアプリケーションを周期実行するため比較的容易に実装できる特徴がある．その一方で，アプリケーション内に周期の長いタスクと短いタスクが混在すると，アプリケーションの実行周期を周期の短いタスクに合わせて設定する必要があり，PShEDアルゴリズムや本論文で対象とした時間保護アルゴリズムのようにデッドラインを用いる手法に比べて，デッドラインに余裕があるタスクの切替え回数が増加する問題がある．また，タスクのデッドライン情報が得られる場合でも，これを有効に活用する手段がない．

6. まとめ

本論文では，複数の組込みリアルタイムアプリケーションが動作するシングルプロセッサシステム向けに，アプリケーション毎のプロセッサ時間を保護するための効率的な時間保護スケジューリングアルゴリズムを提案した．提案アルゴリズムでは，プロセッサ時間と，そのプロセッサ時間を使用するタスクの優先度を関連付けて管理し，不要なタイミングでのプロセッサ時間割当てを減らす．その結果，アプリケーション切り替え回数を削減できることを簡単なアプリケーションを用いて示した．今後は，既存のリアルタイムOSへの実装し，より複雑なアプリケーションを対象として，アプリケーション切替え回数やバジェットリスト管理のための処理時間を計測し，提案アルゴリズムを評価する予定である．

文献

[1] 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，Vol. 48, No. SIG 8(ACS18)，情報処理学会論文誌コンピューティングシステム，pp. 192–202 (2007).

[2] 松原豊，本田晋也，冨山宏之，高田広章：OSEKアプリケーション統合のための柔軟なスケジューリングフレームワーク，Vol. 2007, No. 8，情報処理学会組込みシステムシンポジウム論文集，pp. 33–41 (2007).

[3] Lipari, G. and Baruah, K.: Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems, *Proc. IEEE Real-Time Technology and Applications Symposium* (2000).

[4] Lipari, G. and Buttazzo, G.: Scheduling Real-Time Multi-task Applications in an Open System, *Proc. IEEE 11th Euromicro Conference on Real-Time Systems* (1999).

[5] Lipari, G., Carpenter, J. and Baruah, S.: A framework for acheving inter-application isolation in multiprogrammed, hard real-time environments, Proc. IEEE Real-time System Symposium, pp. 217–226 (2000).

[6] Lipari, G. and Bini, E.: A methodology for designing hierarchical scheduling systems, Journal fo Embedded Computing, Cambridge International Science Publishing (2003).

[7] Deng, Z., J.W.-S.Liu, Zhang, L., Mouna, S. and Frei, A.: An Open Environment for Real-Time Applications, Vol. 16, Real-Time Systems Journal, pp. 155–185 (1999).