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

割込み処理を含むリアルタイムアプリケーション統合のための階層型スケジューリング

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スケジューリング手法として，階層型スケジューリングが提案されている．本論文では，タスクのみを対象とした階層型スケジューリングに対して，割込み処理のスケジューリングを組み合わせることで，タスクと割込み処理の両方で構成されるアプリケーションに対応する手法を検討する．まず，割込み処理のスケジューリングとして，システム全体で行うグローバル固定優先度スケジューリングと，個々のアプリケーション内で行うローカル固定優先度スケジューリングの 2 方式を提案する．次に，それぞれの方式の最大応答時間を解析する手法と，既存の階層型スケジューラシステムに実装するための設計を述べる．最後に，実装・評価を通して，それぞれの方式の有効性を議論する．

Hierarchical Scheduling for Integrating Real-time Applications with Interrupt Routines

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

Many hierarchical scheduling algorithms have been studied for integrating multiple independently developed applications. These algorithms target to real-time applications that consist of only tasks. In this paper, we present two scheduling algorithms for interrupt routines, a global fixed priority scheduling (G-FPS) and a local fixed priority scheduling (L-FPS). In the G-FPS, priority of an interrupt routine is compared among all interrupt routines in the system. On the other hands, in the L-FPS, the priority is compared within an application. We analyze the worst-case response time in both algorithms, and show the design for implementation of them to a real-time OS with a hierarchical scheduling. Finally, we discuss and evaluate proposed algorithms through the result of the implementation.

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・検証されたリアルタイムアプリケーションを，ひとつのプロセッサ上に統合する手法として，2 レベル階層型スケジューラが注目されている．2 レベル階層型スケジューラは，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 階層に配置したスケジューラである．

階層型スケジューラ上で動作するアプリケーションには，システム設計段階でプロセッサ利用率を設定し，システム動作中にはそのプロセッサ利用率に応じてプロセッサ時間（これをバジェットと呼ぶ）が割り当てる．各アプリケーションの実行時間の上限を，割り当てられたバジェットに制限することで，アプリケーション間のプロセッサ時間保護を実現し，アプリケーション統合を容易に実現することを目標とする．

実システムのリアルタイムアプリケーションは，タスクと，主に外部イベントにより起動する処理（これを割込み処理と呼ぶ）で構成されることが多い．しかしながら，これまで提案された階層型スケジューラは，タスクのみで構成されるアプリケーションを対象としており，割込み処理のスケジューリングについて，ほとんど言及されていない．一般に，割込み処理は，タスクより高い優先度が割り当てられ，かつその処理時間も短いため，タスクのスケジューリングにのみ着目する場合には，処理時間を無視することを前提とする場合が多い．その一方で，割込み要因の種類や割込み回数が多い割込み処理を含むアプリケーションを統合する場合には，それらの処理時間を無視できない．例えば，プロセッサの過負荷状態で，割込み処理がプロセッサを占有した時間を無視すると，別のアプリケーションが割り当てられたバジェットを使い切ることができない可能性がある．その結果，アプリケーション間のプロセッサ時間の保護機能が破綻してしまう．

本論文では，これまでに我々が提案した階層型スケジューリングアルゴリズムに対して，割込み処理のスケジューリングを組み合わせ，タスクと割込み処理で構成されるリアルタ

†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

イムアプリケーションも統合できる手法を検討する．具体的には，まず，割込み処理のスケジューリングにおける優先度比較を，システム全体を対象とするグローバル固定優先度スケジューリングと，個々のアプリケーション内の処理を対象とするローカル固定優先度スケジューリングの 2 つに整理する．グローバル固定優先度スケジューリングは，割込み処理をアプリケーションから分離し，システム上のすべてのタスクに優先してスケジューリングする．一般的な割込みアーキテクチャに対して実装が容易であるという利点がある．それに対して，ローカル固定優先度スケジューリングは，タスクと同様に，割込み処理もアプリケーションごとにスケジューリングする．この方式では，割込み処理が必ずタスクに優先して処理されるとは限らないため，実装することが困難である．そこで，割込み処理を，デバイスに依存し非常に短い時間で完了する処理（これを割込みハンドラと呼ぶ）と，割込みに対するサービス処理（これを割込みサービスタスクと呼ぶ）に分割するモデルを導入する．さらに，それぞれのスケジューリングにおける割込み処理の最大応答時間を解析し，プロトタイプの実装による性能評価を通して，それらの有効性を明らかにする．

以下，本論文の構成を示す．第 2 章で関連研究について述べる．第 3 章では，対象とする階層型スケジューラの構成とタスクスケジューリングについて述べる．第 4 章では，割込み処理のスケジューリングについて述べる．第 5 章では，各スケジューリングにおける割込み処理の最大応答時間を解析する．第 6 章で，既存の階層型スケジューラへ実装するための設計を述べ，性能評価を通じて，各スケジューリングの有効性を議論する．最後に，第 7 章で，結論と今後の課題を述べる．

2. 関 連 研 究

これまで数多くの階層型スケジューリングアルゴリズムが提案されている．Deng らは，リアルタイムアプリケーションの統合を想定した階層型スケジューラである Open System を提案した[3),4)]．Open System は，統合するアプリケーションに属するタスクのすべてのリリース時刻と最悪実行時間（WCET）が分かることを前提としている．G. Lipari らは，Open System の適用条件を，すべてのタスクのデッドラインが分かることに，緩めた BSS アルゴリズム[5),6)] と，それを改良した PShED アルゴリズム[7)] を提案している．BSS アルゴリズムを実装したリアルタイムカーネルとしては，*S.Ha.R.K*[8)–10)] がある．

また，我々は，BSS や PShED アルゴリズムにおいては，QoS 制御されたタスクが含まれるアプリケーションを統合する場合に，統合後のプロセッサにおいてデッドラインを満たすことを保証できない場合あることを指摘して，時間保護アルゴリズムを提案した[1)]．さらに，アプリケーションに要求される時間要件や，統合段階で既知であるパラメータが異なるアプリケーションごとに，柔軟にスケジューリングアルゴリズムを選択可能なスケジューリングフレームワークを提案した[2)]．しかしながら，我々の提案も含め，これまで提案された階層型スケジューラは，タスクのみで構成されるアプリケーションを対象としており，割込み処理のスケジューリングについて，ほとんど言及されていない．

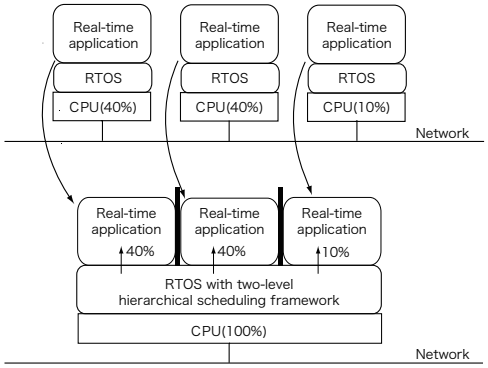

図 1 階層型スケジューラによるアプリケーション統合
Fig. 1 Integrating Real-time applications by an hierarchical scheduler.

リアルタイム OS や汎用 OS を対象とした割込み処理に関する研究としては，これまでいくつかの割込み処理モデルが提案されている[13),14)]．しかしながら，階層型スケジューリングを対象としたものではない．割込み処理のスケジュール可能性解析手法としては,[15)] が提案されている．この手法では，割込み処理とタスクが周期的に発生することを前提としており，そのまま階層型スケジューラでの解析に利用することはできない．

3. アプリケーション統合のための階層型スケジューリング

3.1 アプリケーション統合

本研究では，図 1 に示すように，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目的としている．

実システムにおいては，アプリケーション間のネットワーク通信が存在こともあるが，本論文では，アプリケーションは互いに独立であり，アプリケーション間通信はないことを前

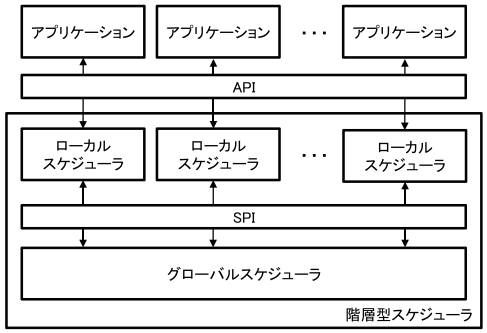

図 2 階層型スケジューラの構成
Fig. 2 Construction of the hierarchical scheduler.

提とする．各アプリケーションは，外部イベントにより起動する割込み処理と，タスクで構成される．アプリケーションに属するすべての処理は，個別プロセッサで固定優先度のプリエンプティブスケジューリングによりスケジュール可能であることを前提とする．さらに，割込み処理の優先度は，同じアプリケーションに属するすべてのタスクの優先度よりも高いものとする．ただし，この前提は，アプリケーション間では必ずしも成立しない．

3.2 階層型スケジューラの概要

ここでは，我々が開発を進めている階層型スケジューラ[2] について述べる．階層型スケジューラの構成を，図 2 に示す．各アプリケーションと階層型スケジューラ（もしくはリアルタイム OS）のインタフェースを API（Application Program Interface）と呼ぶ．階層型スケジューラは，それぞれのアプリケーションに対応するローカルスケジューラと，システムで一つあるグローバルスケジューラで構成され，スケジューラ間のインタフェースである SPI（Scheduler Program Interface）を規定している．ここでは，アプリケーションに属するタスクをスケジュールする機能に絞って説明する．

ローカルスケジューラの主な機能は，次の 3 つである．

- タスクのスケジューリング
  アプリケーションに属するタスクを，静的優先度割当ての固定優先度プリエンプティブスケジューリングによりスケジュールする．実行可能なタスクの中で，最も優先順位の高いタスクの ID 番号を SPI を介してグローバルスケジューラに通知する．
- バジェットの要求
  アプリケーションがバジェットを獲得するために，グローバルスケジューラに対して，バジェットの有効時刻（これを平衡時刻と呼ぶ）を通知する．保護の観点から，アプリケーションのバジェットは，アプリケーション（ローカルスケジューラ）自身で決定することはできない仕組みとしている．
- アプリケーションデッドラインの通知
  ローカルスケジューラは，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を通知する．アプリケーションのデッドラインは，グローバルスケジューラがアプリケーションをスケジューリングする指標となる．

グローバルスケジューラの主な機能は，次に 2 つである．

- アプリケーションのスケジューリング
  アプリケーションを EDF（Earliest Deadline First）スケジューリングでスケジュールし，もっともデッドラインの早いアプリケーションの最高優先順位タスクをプロセッサで実行する．
- アプリケーションへのバジェットの割当て
  グローバルスケジューラは，アプリケーションから通知される平衡時刻と，各アプリケーションの統合プロセッサ上でのプロセッサ利用率から，アプリケーションのバジェットを計算して管理する．アプリケーションに属するタスクが実行されると，その時間分のバジェットが減る．アプリケーションのバジェットが 0 になると，次に平衡時刻が通知されるまで実行できない．バジェットの最大時間は，平衡時刻を通知された時点でのシステム時刻から，平衡時刻までの時間幅のうち，アプリケーションのプロセッサ利用率分の時間である．

SPI は，ローカルスケジューラとグローバルスケジューラが上記の機能を実現する中で，スケジューリングに必要となる情報や，スケジューリング結果を通知・参照するための関数群である．詳細は，6 章で述べる．

3.3 アプリケーションのスケジューリング

階層型スケジューラ上のアプリケーションの振る舞いは，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通知する平衡時刻を決定するアルゴリズムに依存する．例えば，一定の時間間隔で一定のバジェットが割り当てられることを期待する場合には，その一定周期をアプリケーションの起動周期と考え，周期ごとにグローバルスケジューラに対して平衡時刻を通知すれば良い．平衡時刻を決定するアルゴリズムは，アプリケーションごとに選択できる．ここでは，表 1 に示す 3 方式を対象とする．先ほどの一定周期ごとにバジェッ

表 1 アプリケーションのスケジューリング
Table 1 Scheduling Algorithms for Applications.

スケジューリング	バジェットの要求タイミング	アプリケーションのデッドライン
重み付きラウンドロビン	システム全体で一定周期	システム全体で設定
固定デッドライン EDF	アプリケーションごとの起動周期	アプリケーションごとの起動周期
変動デッドライン EDF	実行時にアプリケーションごとに決定	実行時にアプリケーションごとに決定

トを要求するアルゴリズムの場合，システム全体でアプリケーションの起動周期を統一すると重み付きラウンドロビン方式に該当し，アプリケーションごとに異なる起動周期を設定する場合には，固定デッドライン EDF に該当する．固定デッドライン EDF では，アプリケーションの起動周期は一定であり，かつ実行中にも変更できない．アプリケーションのタスクセットから，すべてのタスクをスケジュール可能な，アプリケーションの起動周期とプロセッサ利用率を計算する手法[11),12)] は，固定デッドライン EDF に該当する．それに対して，変更デッドライン EDF は，実行中にローカルスケジューラがアプリケーションのデッドラインを決定する方法である．アプリケーション内の実行可能タスクのデッドラインの中でもっとも早いデッドラインをアプリケーションのデッドラインとする BSS アルゴリズム[5),6)] や，タスクの起動時刻とデッドラインの中でもっとも早いデッドラインをアプリケーションのデッドラインとする時間保護アルゴリズム[1)] の場合は，この方式に該当する．アプリケーションがどの方式により平衡時刻やデッドラインを通知しても，グローバルスケジューラの動作は同じである．

4. 割込み処理のスケジューリング

4.1 スケジューリングの概要

これまで提案された階層型スケジューラは，タスクのみをスケジューリングの対象としているものが多く，割込み処理のスケジューリングについては，ほとんど言及されていない．アプリケーション統合において，割込み処理を含むアプリケーションを対象とする場合には，割込み処理のスケジューリングを決定する必要がある．一般に，割込み処理の実行時間は，タスクに比べて短いことが多く，スケジュール可能性解析においては，それを無視できる前提としても，差し支えない場合もある．その一方で，割込み処理の種類が多い場合や，種類は少なくても割込み回数が非常に多い場合では，無視する実行時間が積載すると，合計時間としては無視し難い場合もある．特に，プロセッサの過負荷状態では，アプリケーションのバジェット管理において割込み処理の実行時間を無視すると，その積み重ねにより，理論上，各アプリケーションは割り当てられたバジェットを必ず使い切れると保証しても，実行時にはバジェットを使い切れない場合がある．その結果，アプリケーション間のプロセッサ時間の保護機能が破綻し，アプリケーション自身には問題がないにも関わらず，別アプリケーションの影響でデッドラインをミスしまう．

本論文では，割込み処理のスケジューリングとして，グローバル固定優先度スケジューリングとローカル固定優先度スケジューリングの 2 方式を提案し，タスクのみを対象とした階層型スケジューラと組合せることで，システム全体として，割込み処理を含むアプリケーションの統合を実現することを検討する．

割込み処理は，割込みを発生する外部イベント（これを割込み要因と呼ぶ）に対応する処理であり，その割込み要因の重要性に応じて割込み処理の優先度が決定される．本論文では，割込み処理のパラメータとして，優先度と，割込み発生時刻からのデッドラインが相対時刻で与えられているものとし，これらは実行中に変更されないものとする．

4.2 グローバル固定優先度スケジューリング

グローバル固定優先度スケジューリングは，割込みが発生した際に，その割込み要因に対応する割込み処理を起動するかを判断するために，割込み処理の優先度をシステム全体を対象に比較する方式である．すなわち，割込み処理の優先度が，システムで実行中の処理の優先度より高い場合に，実行中の処理に優先して割込み処理を実行する．逆に，優先度が同じか低い場合には，割込み要求は受け付けず，割込み処理も起動しない．

グローバル固定優先度スケジューリングでは，優先度が高ければ，実行中の処理がどのアプリケーションに属するものであってもプロセッサを横取りできる．それに対して，タスクは，属するアプリケーションの内部でもっとも優先度が高く，かつアプリケーションのデッドラインが，他のアプリケーションより早い場合に実行される．したがって，システム全体のスケジューリングは，割込み処理は固定優先度スケジューリング，タスクは EDF ベースの階層型スケジューリングという組合せになる．この方式は，一般的なリアルタイム OS やハードウェアアーキテクチャ（プロセッサ，割込みコントローラ）に対して親和性が高く，実装が比較的容易であるという利点がある．その一方で，割込み処理に対する優先度は，システム全体に対して有効になるため，システム設計段階において，すべてのアプリケーションとの優先度関係を考慮する必要がある．したがって，単独で開発，動作検証したアプリケーションを，その詳細な設計を把握せずに統合することは困難である．

4.3 ローカル固定優先度スケジューリング

ローカル固定優先度スケジューリングは，割込み処理のスケジューリングにおける優先度

比較の対象を，属するアプリケーション内に限定する方式である．すなわち，割込みが発生した際に，その割込み要因に対応する処理の優先度が，属するアプリケーション内でもっとも高い場合にのみ割込み処理を起動する．それ以外の場合には，割込み要求は受け付けず，割込み処理も起動しない．

ローカル固定優先度スケジューリングでは，割込み処理起動時の処理は，タスク起動時と同様となる．例えば，変動デッドライン EDF でアプリケーションがスケジュールされる場合，アプリケーションのデッドラインは，属する処理のデッドラインの中でもっとも早いデッドラインに一致する．割込み処理起動時もこれに従い，割込み処理のデッドラインがアプリケーション内でもっとも早いデッドラインとなる場合には，アプリケーションデッドラインを更新する．アプリケーションが再スケジュールされ，もっともデッドラインが早いアプリケーションとなれば，割込み処理が実行される．しかし，別のアプリケーションの方がデッドラインが早ければ，割込み処理は待たされる．ローカル固定優先度スケジューリングでは，割込み処理は，アプリケーションに属するタスクと同様に扱われる．したがって，システム全体のスケジューリングは，割込み処理を含め階層型スケジューリングで統一できる．

この方式は，グローバル固定優先度スケジューリングとは逆に，割込み処理に対する優先度はアプリケーション内部でのみ有効であるため，各アプリケーションの設計段階で他のアプリケーションの設計を考慮する必要がなく，個別に開発，動作検証したアプリケーションを統合する場合には有効であると思われる．その一方で，本来，高い応答性が要求される割込み処理が，別アプリケーションのタスクに待たされる可能性があり，一般的なリアルタイム OS やハードウェアアーキテクチャでの割込み処理スケジューリングとは異なる．そのため，OS やハードウェアの割込みアーキテクチャを変更せず，効率的に実装することは困難である．

5. 解　　析

5.1 用語の定義

ここでは，割込み処理スケジューリングを実装する前に，割込み処理の最大応答時間を解析するための用語を定義する．統合プロセッサでは，N 個のアプリケーションが動作し，それぞれのアプリケーションは，A_i= (U_i,D_i) で表す．ここに，U_i は A_i に設定されるプロセッサ利用率，D_i はアプリケーションの相対デッドラインである．重み付きラウンドロビンの場合には，すべてのアプリケーション周期を P とすると，各アプリケーションの相対デッドラインも P に一致する．アプリケーションが変動デッドライン EDF でスケジュールされる場合には，D_i も実行時に変動する．

A_i は，処理単位の集合で，$\tau_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$ で表される．処理単位 τ_{ij} は，割込み処理かタスクのいずれかで，(p_{ij}, d_{ij}, c_{ij}) で表す．ここに，p_{ij} は優先度，d_{ij} は相対デッドライン，c_{ij} は統合前の個別プロセッサにおける最大実行時間である．最大実行時間の情報は，実行時のスケジューリングでは使用せず，解析時のみ必要な情報である．τ_{ij} は，p_{ij} が高い順に整列し，$p_{ij} \geq p_{ij+1}$ が満たされるものとする．なお，c_{ij} は，個別プロセッサでの最大実行時間なので，統合プロセッサは，$c_{ij} * U_i$ となる．

5.2 最大応答時間解析

割込みスケジューリングごとに，割込み処理の最大応答時間を解析する．まず，グローバル固定優先度スケジューリングにおいて，割込み処理 τ_{ij} の応答時間 R_{ij} が最大となるのは，その Critical Instant である．すなわち，すべてのアプリケーションに属する割込み処理のうち，τ_{ij} と同じか，より高い優先度をもつ処理が同時にすべて起動する状況である．よって，最大応答時間は，以下の漸化式で計算できる．

$$R_{ij} = \sum_{k=1}^{N} \sum_{\forall \tau_{kl} \in hp_k(l)} \left\lceil \frac{R_{ij}}{t_{kl}} \right\rceil * c_{kl} + c_{ij} \quad (1)$$

ここに，$hp_k(l)$ は，アプリケーション A_k に属し，かつ l 以上の優先度をもつ処理単位の集合である．また，R_{ij} の初期値を c_{ij} として計算し，計算結果が 2 回同じ値になるまでに再帰的に計算する．

次に，ローカル固定優先度スケジューリングの最大応答時間を解析する．ローカル固定優先度スケジューリングにおいて，応答時間がもっとも長くなるのは，アプリケーションのバジェットを使い切った時点で割込み要求が発生し，かつ次のアプリケーションのデッドラインまでの時間の一番最後にスケジュールされる状況である．割込み処理がが属するアプリケーションのデッドラインは，スケジューリング方式により異なるため，それぞれのスケジューリングに対して解析する．まず，重み付きラウンドロビンの場合は，以下のようになる．

$$R_{ij} = \frac{2P}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (2)$$

第 1 項は，他のアプリケーションの最大実行時間，第 2 項は τ_{ij} と同じアプリケーション

に属する割込み処理で，同じかより高い優先度をもつ割込み処理の実行時間である．同様に，固定デッドライン EDF の場合は，第 1 項の他のアプリケーションの最大実行時間が異なり，次のようになる．

$$R_{ij} = \frac{2 * D_i}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (3)$$

変動デッドライン方式の場合も同様に，第 1 項の他のアプリケーションの最大実行時間が異なり，次のようになる．

$$R_{ij} = \frac{max_k\{d_{ik}\} + d_i j}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (4)$$

以上より，多くの場合，割込み処理の最大応答時間は，ローカル固定優先度に比較して，別アプリケーションの実行により待たされることがないグローバル固定優先度スケジューリングの方が短い．ただし，実際には，アプリケーションの構成に依存するために，これらの解析手法を用いて，比較する必要がある．

6. 実装と評価

6.1 実 装 環 境

割込み処理のスケジュール方式の実装容易性と，割込み処理の応答性能を評価するため，開発中のプロトタイプシステムにそれぞれの方式を実装する．実装ターゲットボードは，オークス電子社製の OAKS32R ボードである．このボードには，M32R プロセッサ（動作周波数は 66MHz，キャッシュは OFF）を搭載し，メモリは，64K バイトのコア内蔵 SRAM と，8M バイトの外部 SDRAM を持つ．今回は，すべて外部 SDRAM 上に配置する．

実装のベースとするシステムは，オープンソースの TOPPERS/ASP カーネルを階層型スケジューラに拡張したプロトタイプシステムである．このプロトタイプシステムの基本性能を表 2 に示す．

6.2 割込み処理の流れ

次に，割込み処理の流れについて述べる．割込み処理は，プロセッサや割込みコントローラ（IRC）のアーキテクチャに依存するため，本論文では，TOPPERS/ASP カーネルで規定されている標準割込み処理モデル[16] を前提とする．これを．図 3 に示す．この図では，すべてをハードウェアで実現されていると想定して描かれている．周辺デバイスからの割込み要求が発生すると，割込みコントローラ（IRC）を経由して，プロセッサに伝えられる．プロセッサは，NMI 以外の割込み要求は，以下の 4 つの条件を満たす場合に割込み要求を受け付ける．

(1) 割込み要求禁止フラグがクリアされていること
(2) 割込み要求の持つ割込み優先度が，現在の割込み優先度マスクの現在値よりも高いこと

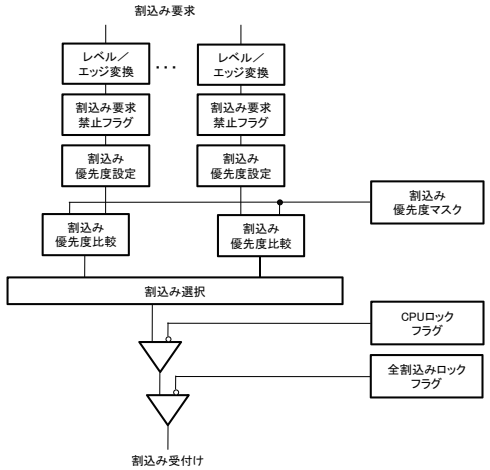

図 3 割込み処理モデル
Fig. 3 Interrupt Processing Model.

表 2 プロトタイプシステムの基本性能
Table 2 Processing Time of Basic Functions in the Prototype System.

処理内容	命令名称	実行時間（us）
タスク起動（タスク切り替え有り）	act_tsk	62
タスク起動（タスク切り替えなし）	act_tsk	5
タスク起動（非タスクコンテキスト）	iact_tsk	4
システム時刻更新とバジェット管理	signal_time	10
バジェット計測開始	budget_timer_start	6
バジェット計測停止	budget_timer_stop	7
アプリケーションの最高優先順位タスク ID の通知（SPI）	set_schedtsk	5
アプリケーションの平衡時刻の通知（SPI）	set_balancing_time	10
アプリケーションの絶対デッドラインの通知（SPI）	set_deadline	1

(3)　全割込みロックフラグがクリアされていること
(4)　CPU ロックフラグがクリアされていること

各フラグはハードウェアで実現できるプロセッサもあれば，ソフトウェアにより実現しなければならないプロセッサもある．M32R では，CPU ロックフラグ（カーネル管理の割込みの受付を禁止する）のみソフトウェアにより実現する．割込み要求が受け付けられた後の処理は，割込み処理のスケジューリングにより異なるが，基本的には次の処理が実行される．詳細は，次節にて述べる．

(1)　割込み入口処理
(2)　割込みハンドラ
(3)　割込みサービスルーチン（グローバル固定優先度方式の場合）
(4)　割込みサービスタスクの起動（ローカル固定優先度方式の場合）
(5)　割込み出口処理

6.3 実装方法

割込み要求を受け付け後の処理を，割込み処理のスケジューリングごとに説明する．まず，グローバル固定優先度スケジューリングにおける処理を図 4 に示す．グローバル固定優先度スケジューリングは，割込みが発生した際に，その割込みに対応する処理の優先度が，実行中の処理の優先度より高い場合に，実行中の処理に優先して割込み処理を実行する．この場合は，先の割込み処理モデルをそのまま用いることができ，割込み要求受け付け後に，割込みハンドラから，アプリケーションが登録した割込みサービスルーチンを実行することで実現する．割込み応答時間は，割込み要求受付けから，割込みーサービスルーチンを起動するまでの時間とする．また，割込みサービスルーチンの実行時間が非常に短いアプリケーションにおいて，割込みサービスルーチンの実行時間を階層型スケジューラのバジェット管理の対象から外す（無視する）場合には，割込みサービスルーチンの実行前後にあるバジェット計測操作は不要となる．

次に，ローカル固定優先度スケジューリングにおける処理を図 4 に示す．ローカル固定優先度スケジューリングは，割込み処理が属するアプリケーション内で優先度を比較する必要がある．まず，割込み要求発生後，カーネル処理などの割込み禁止区間（割込みロックされている状態）でない場合を除いては，一端割込み要求を受け付ける．その後で，割込み要求に対応する割込み処理用のタスク（これを割込みサービスタスクと呼ぶ）を起動する．なお，タスクを起動することは，即座に実行することを意味するのではなく，アプリケーション内での優先度スケジューリングを実行することを意味する．割込みサービスルーチンは，

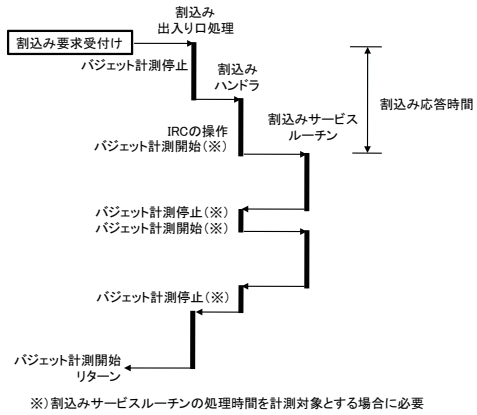

図 4　グローバル固定優先度スケジューリングにおける処理
Fig. 4　Interrupt Processing Procedure in G-FPS.

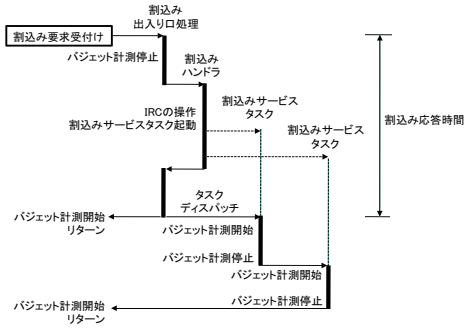

図 5　ローカル固定優先度スケジューリングにおける処理
Fig. 5　Interrupt Processing Procedure in L-FPS.

呼び出されると即座に実行されるのに対して，割込みサービスタスクは，起動直後に実行されない．すなわち，起動後に，アプリケーションのデッドラインを更新し，アプリケーションのデッドラインがもっとも早ければ，実行中の処理から割込みサービスタスクに実行を切り替えるディスパッチ要求を出す．ディスパッチ要求が発生しているアプリケーションでは，割込み出口処理で，割込みサービスタスクに実行が切り替わる．割込み処理をタスク化している理由は，アプリケーション内に限定した優先度スケジューリングを容易に実現できることと，アプリケーション間を EDF でスケジュールすることで，割込み処理実行中であっても別のアプリケーションに属するタスクを実行できるよう実行状態を保存する必要があるためである．

6.4 性能評価

割込みスケジューリングごとに，割込み応答時間を測定した結果を表 3 に示す．測定した応答時間は，割込み要求が発生してから，アプリケーションの割込み処理が起動されるまでの時間である．今回は，それぞれの割込み処理はシステム内で最高優先度をもつものとし，他のアプリケーションや割込み処理に実行を邪魔される時間は含めていない．すなわち，割込要求の発生に対して，もっとも早く応答できる状況を想定している．

割込みハンドラは，割込み処理スケジューリングに関係なく，最初に起動するハンドラで

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

表 3 割込み処理の応答時間
Table 3 Reponse Time of Interrupt Routines.

割込み処理	実行時間（μ s）
割込みハンドラ	3
割込みサービスルーチン（グローバル固定優先度）	10 (3)
割込みサービスタスク（ローカル固定優先度）	37

表 4 割込み処理のスケジューリングの比較
Table 4 Comparison between the Scheduling Algorithms.

スケジューリング	優先度比較の範囲	階層型スケジューラへの適用性	実装容易性	割込み応答性
グローバル固定優先度	システム全体	×	○	○
ローカル固定優先度	アプリケーション内	○	×	△

ある．グローバル固定優先度スケジューリングにおける割込みサービスルーチンは，割込みハンドラに対して割込みサービスルーチン呼び出しと，バジェット管理処理を加えて 10 μ秒となった．バジェット計測処理がない場合には，3 μ秒と割込みハンドラとほぼ同程度の割込み応答時間である．一方，ローカル固定優先度スケジューリングの割込みサービスタスクの応答時間は，割込みハンドラ中に起動されてから割込み出口処理からタスク切り替えが発生するまで実行を待たされるため，割込みサービスルーチンの応答時間に比べて 4 倍弱長くなった．より具体的には，アプリケーション内でのタスクスケジューリング，アプリケーションのデッドライン更新，アプリケーションのスケジューリングなどの処理時間が応答時間増加の要因となっている．これらの処理の一部をハードウェアで実現することで，ローカル固定優先度スケジューリングの割込み応答時間を短縮できると思われるが，それは今後の課題とする．

6.5 議　　論

最後に性能評価結果を踏まえ，割込み処理スケジューリングを表 4 に整理し，有用性を議論する．グローバル固定優先度スケジューリングは，割込み処理をアプリケーションから分離し，システム上のすべてのタスクに優先してスケジューリングする．割込み処理モデルに対して実装が容易であり，割込み応答性も高い．それに対して，ローカル固定優先度スケジューリングは，タスクと同様に，割込み処理もアプリケーションごとにスケジューリングする．既存の割込み処理モデルに対しては，実装は困難で，割込み処理をタスク化した割込みサービスタスクによる実装では，割込み応答時間は，割込みサービスタスクの 10 μ秒に対して 37 μ秒に増加した．階層型スケジューラへの適用性は，割込み処理も階層型スケジューラでそのまま扱えるという意味で，ローカル固定優先度スケジューリングの方が高い．このことは，アプリケーションのスケジュール可能性解析において，これまでの解析手法を利用できるという利点がある．それに対して，グローバル固定優先度スケジューリングは，割込み処理とタスクのスケジューリングが異なるため，アプリケーションのスケジュール可能性解析においては，これまでの解析手法をそのまま利用することができない．以上より，既存の割込みアーキテクチャにおいて，容易に実現する場合には，グローバル固定優先度スケジューリングの方が有効であるが，ローカル固定優先度助中 リングおける実装方法と割込み応答性を改善できれば，階層型スケジューラにおいては，ローカル固定優先度スケジューリングの方が適すると考えられる．

グローバル固定優先度スケジューリングにおけるスケジュール可能性解析手法としては，例えば，割込み処理を RMA で，アプリケーションを EDF で解析する手法[17] を適用することが考えられる．しかしながら，アプリケーションに属する個々のタスクがスケジュール可能であることを解析するには，さらに検討が必要であると思われるが，これは今後の課題とする．

7. おわりに

本論文では，タスクのみを対象とした階層型スケジューリングに対して，割込み処理のスケジューリングを組み合わせることで，タスクと割込み処理で構成されるアプリケーションに対応する手法を提案した．まず，割込み処理のスケジューリングをシステム全体で行うグローバル固定優先度方式と，個々のアプリケーション内で行うローカル固定優先度方式の二つに整理した．次に，それぞれの方式を階層型スケジューラで実現するための割込み処理モデルについて述べた．最後に，それぞれの割込み処理モデルを実装・評価した結果，アプリケーション統合への適用性の高いローカル固定優先度方式は，実装が比較的容易なグローバル固定優先度方式に比較して，30 μ秒程度オーバヘッド多いことが明らかになった．今後は，グローバル固定優先度方式におけるスケジュール可能性解析手法と，ローカル固定優先度方式のオーバヘッドを削減するために，割込み処理のハードウェア化を検討する．

参 考 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，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).

2) 松原豊，本田晋也，冨山宏之，高田広章：リアルタイム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，情報処理学会論文誌，Vol.49, No.10, pp.3508-3518 (2008).
3) Z.Deng and J.W.-S.Liu and J.Sun, *A Scheme for Scheduling Hard Real-Time Applications in Open System Environment*, In Proceedings of 9th Euromicro Workshop on Real-Time Systems, pp. 191-199 (1997).
4) 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).
5) 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).
6) G.Lipari and G.Buttazzo, *Scheduling Real-Time Multi-task Applications in an Open System*, In Proceedings of IEEE 11th Euromicro Conference on Real-Time Systems (1999).
7) 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).
8) P. Gai and L.Abeni and M.Giorgi and G.Buttazzo, *A New Kernel Approach for Modular Real-Time systems Development*, In Proceedings of IEEE 13th Euromicro Conference on Real-Time Systems, IEEE Computer Society Press (2001).
9) G.Lipari and P.Gai and M.Trimarchi and G.Guidi and P.Ancilotti, *A Hierarchical Framework for Component-Based Real-Time Systems*, In Proceedings of IEEE International Symposium on Component-based Software Engineering, IEEE Computer Society Press. , (2004).
10) P.Gai and G.Lipari and L.Abeni and M.diNatale and E.Bini, *Architecture for A Portable Open Source Real Time Kernel Environment*, In Proceedings of 2nd Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial, (2000).
11) G.Lipari and E.Bini, *A methodology for designing hierarchical scheduling systems*, Journal of Embedded Computing, Cenbridge International Science Publishing, (2003).
12) I.Shin and I.Lee, *Compositional Real-time Scheduling Framework*, In Proceedings of IEEE Real-time Systems Symposium, pp.57-67 (2004).
13) L. E. Leyva-del-Foyo and P. Mejia-Alvarez and D. de Niz, *Predictable Interrupt Scheduling with Low Overhead for Real-Time Kernels*, In Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing System and Applications, pp.14-23 (2006).
14) Y.Zhang and R.West, *Process-Aware Interrupt Scheduling and Accounting*, In Proceedings of the 27th IEEE Real-Time Systems Symposium, pp. 191-201 (2006).
15) K.Jeffay and D.L.Stone, *Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems*, In Proceedings of the 14th IEEE Real-Time Systems Symposium, pp.212-221, (1993).
16) TOPPERS プロジェクト：TOPPERS 新世代カーネル統合仕様書 Release 1.1.0, (2009).
17) C.L.Liu and James W.Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of the ACM (JACM), v.20 n.1, p.46-61, (1973).

○正誤文章
【元の文章】

5. 解　　析

5.1 用語の定義の第 2 パラグラフ

A_i は，処理単位の集合で，$\tau_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$ で表される．処理単位 τ_{ij} は，割込み処理かタスクのいずれかで，(p_{ij}, d_{ij}, c_{ij}) で表す．ここに，p_{ij} は優先度，d_{ij} は相対デッドライン，c_{ij} は統合前の個別プロセッサにおける最大実行時間である．最大実行時間の情報は，実行時のスケジューリングでは使用せず，解析時のみ必要な情報である．τ_{ij} は，p_{ij} が高い順に整列し，$p_{ij} \geq p_{ij+1}$ が満たされるものとする．なお，c_{ij} は，個別プロセッサでの最大実行時間なので，統合プロセッサは，$c_{ij} * U_i$ となる．

5.2 最大応答時間解析

割込みスケジューリングごとに，割込み処理の最大応答時間を解析する．まず，グローバル固定優先度スケジューリングにおいて，割込み処理 τ_{ij} の応答時間 R_{ij} が最大となるのは，その Critical Instant である．すなわち，すべてのアプリケーションに属する割込み処理のうち，τ_{ij} と同じか，より高い優先度をもつ処理が同時にすべて起動する状況である．よって，最大応答時間は，以下の漸化式で計算できる．

$$R_{ij} = \sum_{k=1}^{N} \sum_{\forall \tau_{kl} \in hp_k(l)} \left\lceil \frac{R_{ij}}{t_{kl}} \right\rceil * c_{kl} + c_{ij} \quad (1)$$

ここに，$hp_k(l)$ は，アプリケーション A_k に属し，かつ 1 以上の優先度をもつ処理単位の集合である．また，R_{ij} の初期値を c_{ij} として計算し，計算結果が 2 回同じ値になるまでに再帰的に計算する．

次に，ローカル固定優先度スケジューリングの最大応答時間を解析する．ローカル固定優先度スケジューリングにおいて，応答時間がもっとも長くなるのは，アプリケーションのバジェットを使い切った時点で割込み要求が発生し，かつ次のアプリケーションのデッドラインまでの時間の一番最後にスケジュールされる状況である．割込み処理がが属するアプリケーションのデッドラインは，スケジューリング方式により異なるため，それぞれのスケジューリングに対して解析する．まず，重み付きラウンドロビンの場合は，以下のようになる．

$$R_{ij} = \frac{2P}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (2)$$

第 1 項は，他のアプリケーションの最大実行時間，第 2 項は τ_{ij} と同じアプリケーションに属する割込み処理で，同じかより高い優先度をもつ割込み処理の実行時間である．同様に，固定デッドライン EDF の場合は，第 1 項の他のアプリケーションの最大実行時間が異なり，次のようになる．

$$R_{ij} = \frac{2 * D_i}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (3)$$

変動デッドライン方式の場合も同様に，第 1 項の他のアプリケーションの最大実行時間が異なり，次のようになる．

$$R_{ij} = \frac{max_k\{d_{ik}\} + d_{ij}}{1 - U_i} + \sum_{\forall \tau_{ik} \in hp_i(j)} \left\lceil \frac{R_{ij}}{t_{ik}} \right\rceil * c_{ik} + c_{ij} \quad (4)$$

以上より，多くの場合，割込み処理の最大応答時間は，ローカル固定優先度に比較して，別アプリケーションの実行により待たされることがないグローバル固定優先度スケジューリングの方が短い．ただし，実際には，アプリケーションの構成に依存するために，これらの解析手法を用いて，比較する必要がある．

【正しい文章】

5. 解　　析

5.1 用語の定義の第 2 パラグラフ

A_i は，処理単位の集合で，$\tau_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$ で表される．処理単位 τ_{ij} は，割込み処理かタスクのいずれかで，$(t_{ij}, p_{ij}, d_{ij}, c_{ij})$ で表す．ここに，t_{ij} は起動周期（もしくは，最小到着間隔），p_{ij} は優先度，d_{ij} は相対デッドライン，c_{ij} は統合前の個別プロセッサにおける最大実行時間である．c_{ij} は，個別プロセッサにおける実行時間を示すことから，統合プロセッサでの最大実行時間は，$c_{ij} * U_i$ で計算する．

5.2 最大応答時間解析

割込みスケジューリングごとに，割込み処理の最大応答時間を解析する．まず，グローバル固定優先度スケジューリングにおいて，割込み処理 τ_{ij} の応答時間 R_{ij} が最大となるのは，Critical Instant の場合である．すなわち，すべてのアプリケーションに属する割込み処理のうち，τ_{ij} と同じか，より高い優先度をもつ処理が，すべて同時に起動する状況である．まず，アプリケーション A_k に対して，ある時刻 x までに，τ_{ij} と同じかより優先度の高い処理の合計処理時間を計算する関数 $HP_k(x, \tau_{ij})$ を次のように定義する．

$$HP_k(x, \tau_{ij}) = \sum_{\forall \tau_{kl} \in hp_k(p_{ij})} \left\lceil \frac{x}{t_{kl}} \right\rceil * c_{kl} * U_k \tag{1}$$

ここに，$hp_k(p_{ij})$ は，アプリケーション A_k に属し，p_{ij} と同じか，高い優先度をもつ処理単位の集合である．単に $HP(x, \tau_{ij})$ とした場合は，τ_{ij} と同じアプリケーションを対象に計算するものとする．この関数を用いると，グローバル固定優先度スケジューリングにおける割込み処理 τ_{ij} の最大応答時間は，次の漸化式で計算できる．

$$R_{ij} = \sum_{k=1}^{N} HP_k(R_{ij}, \tau_{ij}) + c_{ij} * U_i \tag{2}$$

この漸化式において，R_{ij} の初期値を $c_{ij} * U_i$ として計算し，計算結果が 2 回同じ値になるまで再帰的に計算する．

次に，ローカル固定優先度スケジューリングの最大応答時間を解析する．ローカル固定優先度スケジューリングにおいて，応答時間がもっとも長くなるのは，アプリケーションのバジェットを使い切った直後に割込み要求が発生し，かつ，次のアプリケーションのデッドラインまでの時間において，一番最後にスケジュールされる状況である．割込み処理が属するアプリケーションのデッドラインは，スケジューリング方式により異なるため，それぞれのスケジューリングに対して最大応答時間を解析する．

まず，重み付きラウンドロビンの場合は，次のように計算できる．

$$R_{ij} = \left\lceil \frac{HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i}{P * U_i} \right\rceil * P(1 - U_i) + HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i \tag{3}$$

R_{ij} の初期値は，$c_{ij} * U_i$ である．漸化式の第 1 項は，他のアプリケーションが先に実行されることによる遅延時間である．第 2 項は，τ_{ij} と同じアプリケーションに属する割込み処理で，同じかより高い優先度をもつ割込み処理の実行時間である．

固定デッドライン EDF の場合，第 1 項は，バジェットを使い切った直後に割込みが発生することによる遅延時間と，バジェットが割り当てられた後に，他のアプリケーションが先に実行されることによる遅延時間である．

$$R_{ij} = \left(1 + \left\lceil \frac{HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i}{D_i * U_i} \right\rceil \right) * D_i(1 - U_i) + HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i \tag{4}$$

この場合も，R_{ij} の初期値は $c_{ij} * U_i$ である．

変動デッドライン方式の場合は，実行時にアプリケーションのデッドラインが変動することから，応答時間が d_{ij} を越えてしまった後，すなわち，デッドラインをミスした後に設定されるアプリケーションデッドラインを想定する必要がある．ここでは，仮に，τ_{ij} がデッドラインをミスした後も，アプリケーションデッドラインとして，d_{ij} を継続的に用いるものとすると，最大応答時間は，次のようになる．

$$\begin{aligned} R_{ij} = {} & \max_k \{d_{ik}\}(1 - U_i) + \left\lceil \frac{HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i}{d_{ij} * U_i} \right\rceil * d_{ij}(1 - U_i) \\ & + HP(R_{ij}, \tau_{ij}) + c_{ij} * U_i \end{aligned} \tag{5}$$

この場合も，R_{ij} の初期値は $c_{ij} * U_i$ である．第 1 項は，アプリケーション A_i でもっとも長いデッドラインに対するバジェットを使い切った直後に，割込み要求が発生する場合の最大遅延時間である．第 2 項は，デッドライン d_{ij} に対して，アプリケーション A_i が最後にスケジュールされる場合の遅延時間である．

以上より，割込み処理の最大応答時間は，各アプリケーションの相対デッドラインや，処理単位の構成に依存するが，多くの場合，別アプリケーションの実行に待たされることのないグローバル固定優先度スケジューリングの方が，ローカル固定優先度スケジューリングよりも応答時間が短くなると思われる．