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

2008－SLDM－134／2008－EMB－8（35）
2008 / 3 / 28

高精度なロボット制御のための時間管理機構の設計と実装

上山 真生† 水頭 一壽† 山崎 信行†

† 慶應義塾大学大学院 理工学研究科, 〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †ueyama@ny.ics.keio.ac.jp

あらまし 本論文では，ロボット制御に必要なリアルタイム性をOSが保証するための時間管理機構を提案する．まず，周期タスクのデッドラインがデッドラインをミスしないことを保証するために，周期タスク生成時にアドミッションコントロールを行う．従来のリアルタイムOSでは，実行タイミングの予測性が高い静的優先度アルゴリズムが採用されてきたが，理論的に任意のタスクセットについてデッドラインを保証可能な資源利用率が低く，アドミッションコントロールとは相性が悪かった．しかしながら，本リアルタイムOSでは，モータ制御のように実行タイミングジッタを許容しないタスクの実行を，タイマ割込みサービスルーチンに任せる．ジッタを許容しないタスクを周期タスクのスケジューリングから分離したことにより，周期タスクのスケジューリングの際にジッタを考慮しなくてすむため，実行タイミングの予測性は低いが，理論的に保証可能な資源利用率の高い動的優先度アルゴリズムを採用することが可能となる．この時間管理機構により，リアルタイム性を保証しつつ，計算資源を有効に使うことが可能となる．
キーワード ロボット制御，リアルタイムOS，アドミッションコントロール

Time Management Functions for Sophisticated Robot Control

Masao UEYAMA†, Kazutoshi SUITO†, and Nobuyuki YAMASAKI†

† Graduate School of Science and Technology, Keio University, Hiyoshi 3-14-1, Kohoku-ku, Yokohama, Kanagawa, 223-8522, Japan
E-mail: †ueyama@ny.ics.keio.ac.jp

Abstract This paper describes the time management functions of a real-time operating system (RTOS) for sophisticated robot control. A robot motion control consists of hard deadline tasks. Therefore, our RTOS has an admission control function which is called at task creation to ensure meeting these deadlines of all periodic tasks. Our RTOS supports Earliest Deadline First (EDF) scheduling algorithm, due to segregate timing critical tasks from periodic task scheduling by using timer interrupt service routine.
Key words Robotics, RTOS, Admission Control

1. 序 論

ロボット制御システムは，運動学演算や行動計画等の周期を持つタスクから成る．そして，それらの周期タスク同士には深い依存関係があり，必ず次の周期までに処理を終えなければ，システムが破綻してしまう可能性がある．各周期タスクの制限時間のことを，デッドライン[1]と呼ぶ．さらに，アクチュエータへの指令値の書き込みや，センサデータの読み出しを行うタスクには，各周期の実行タイミングにジッタが生じないことが求められる．

これまでのロボット開発では，各ロボットに応じて最適なコントローラ及び制御ソフトウェアを作込むことによって，必要最低限の処理能力で最適なシステムを構築してきた．作り込みは，ファクトリーオートメーションのように，あらかじめタスクが決まっているシステムにおいて，非常に有効であった．しかし，ロボットに求められるタスクは年々複雑かつ多様化しており，これまでのような作込みによる実装が困難になりつつある．

そこで近年，組み込み分野にもART-Linux[5]やTRON[4]系のリアルタイムOSを導入しようという試みがなされている．一般的に，OSを導入するとハードウェアの抽象度が上がるため，プログラミングは容易になるが，時間予測性が低下してしまうという問題がある．ART-LinuxやTRON系のリアルタイムOSでは，カーネルによる遅延を極力排除することにより，システムコール等の応答性を高めている．つまり，タスクの時間管理を行うのはあくまでアプリケーション開発者であり，OS側はなるべく邪魔をしないことで，リアルタイム性を高めようというアプローチであることが分かる．

2. プロトタイプ OS の構成

本研究の目標は，導入するだけで，誰でも手軽に質の高いロボット開発を可能とするリアルタイム OS を開発することである．既存のリアルタイム OS では，タスクスケジューリングのうちの大部分をアプリケーション開発者が管理しなくてはならず，ロボット制御のように高いリアルタイム性が求められるシステムを開発するにあたり，大きな負担となっていた．そこで本研究では，OS 側でタスクのリアルタイム性を保証する機構を設け，ロボット開発の負荷を軽減する．提案するリアルタイム OS の構成を図 1 に示す．

また，組込みという特性上，汎用 PC に使われるような，計算資源が豊富だが消費電力の高いコントローラではなく，計算資源の制約は厳しいが消費電力の低いコントローラにも適応可能でなくてはならない．そのためには，上記の機能をなるべくシンプルに，また，なるべくメモリを必要としないアルゴリズムで実装する必要がある．

以降，本論文で提案する時間管理機構の設計について述べ，それぞれのアルゴリズムは，組込みに採用されるあまり処理能力の高くないプロセッサにおいても，充分実現可能かどうか，検討を行う．本論文では紙面の関係上，非周期サーバについては省略する．

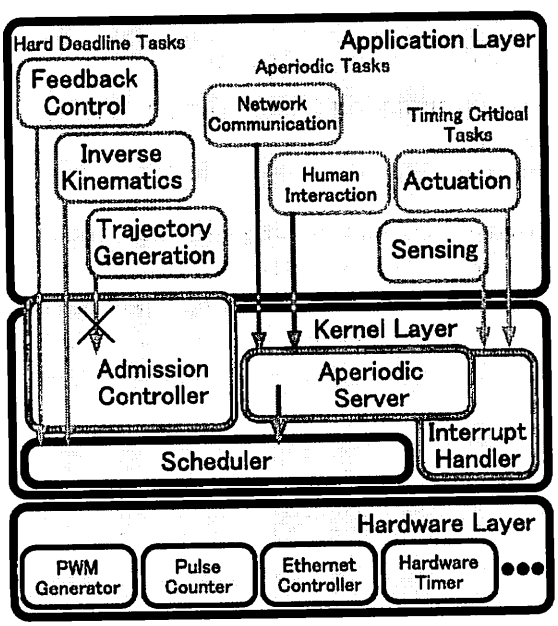

図 1 プロトタイプ OS の概念図

2.1 スケジューラ

本研究で開発をおこなったリアルタイム OS は，静的優先度アルゴリズムである *Rate Monotonic*(RM) と，動的優先度アルゴリズムである *Earliest Deadline First*(EDF) [3] の両方を実装し，比較を行う．

ただし，公平な評価を行うためにはそれぞれのアルゴリズムについて最適化する必要がある．そこで，なるべくスケジューラによるオーバーヘッドを小さくするために，また OS 自体のメモリ量を削減するために，動的なアルゴリズムの変更はサポートせず，コンパイルオプションによって切り替えるものとする．

2.2 タスク制御ブロック

本論文で実装したリアルタイム OS のタスク制御ブロック (TCB) を図 2 に示す．ただし，スケジューリングに直接関係無いメンバ変数は省略してある．TCB は，カーネルが固定配列として用意する方法と，ユーザタスクが利用するスタック領域の先頭に用意する方法がある．固定配列として用意する方法では，一箇所にデータがまとまっているため，操作が簡単である．しかし，想定する全てのタスクの領域をあらかじめ確保しておかなくてはならないため，カーネルが使用するメモリ量が増加してしまう．一方，タスク生成時にユーザスタックの先頭に確保する方法では，現時点で生成されている分のタスクのメモリ領域しか必要としない．そのため，一般的な OS では，一度に数百個のタスクを扱うために，TCB は各タスクが利用するスタック領域の先頭に配置される．しかし，TCB がメモリの至る所に分散し，管理が複雑になってしまうため，頻繁に TCB にアクセスする必要があるリアルタイムスケジューリングには向いていないといえる．また，ロボット制御システムにおいては，せいぜい数十個のタスクしか必要としない．よって本論文では，TCB をカーネルに固定配列を用意する．

このとき，優先度を決定する部分以外のコードを使いまわせるように，priority とリアルタイムパラメータである attr 構造体を分離する．attr 構造体の wcet はそのタスクの最悪実行時間で，アプリケーション開発者が適切な値を設定する必要がある．

```
typedef struct task_struct {
  ulong_t   func;        // 開始アドレス
  ulong_t   pc;          // 復帰アドレス
  ulong_t   priority;    // 優先度
  atomic_t  state;       // 状態
  byte_t    type;        // タイプ
  rt_attr   attr;        // リアルタイム属性
} task_t;

typedef struct realtime_attribute {
  ulong_t   period;      // 周期
  ulong_t   release;     // リリース時間
  ulong_t   absolute;    // 絶対デッドライン
  ulong_t   relative;    // 相対デッドライン
  ulong_t   wcet;        // 最悪実行時間
} rt_attr;
```

図 2 Task Control Block

2.3 スケジューラ呼び出し

スケジューラの呼び出しには，あるタスクが終了したときに起動される明示的呼び出しと，タイマによって一定の間隔で起動される暗黙的呼び出しの二通りがある．また，暗黙的呼び出しが呼び出される間隔のことを *(system) tick* と呼ぶ．

明示的呼び出しはタスクの終了時に呼び出され，次のタスクにコンテキストスイッチする関数である．本論文では，実行中のタスクも RQ に存在するため，明示的呼び出しによって RQ から取り除く．そして次に実行するタスクを RQ から選択する．選択方法については RQ の構造に依存し，後で詳しく述べる．暗黙的呼び出しは，システムの *tick* 毎に呼び出され，新しいタスクがリリースされていないかを調べる関数である．一般的なスケジューラでは，*tick* として数 msec〜数百 msec 程度の固定値が割り当てられ，その都度コンテキストスイッチを行うことで，擬似的なマルチスレッド環境を実現している．そのため，タスクの周期は *tick* の粒度に大きく左右されることになるが，過剰な呼び出しは資源利用率の低下を招いてしまう．そこで本リアルタイム OS では，タスクの周期を維持しつつ計算量をできるだけ削減するために，*tick* をタスクセット中の全タスク周期の最小公倍数に設定する．

2.4 Ready Queue

一般的な OS では，タスク数が増加しても計算量が増加しにくいことから，単純リストを要素に持つ単純リストを要素に持つ優先度配列構造が用いられることが多い．しかし優先度配列構造は，優先度数分の要素を持つ配列を用意しなければならない上に，RQ の管理が複雑であるという欠点がある．一方で優先度リスト構造では，最高優先度タスクは必ずリストの先頭にあるため，明示的なスケジューラ呼び出しのオーバーヘッドが小さい．しかし，タスクのリリース時には優先度に応じて適切な位置に挿入しなくてはならないため，暗黙的なスケジューラ呼び出しの際のオーバーヘッドが大きい．ただし，TCB の所でも述べたように，対象とするロボット制御システムはタスク数が比較的少ないという特徴がある．そこで本論文では，RQ として，より単純で，使用するメモリ量が少ない優先度リスト構造を用いる．また，優先度リストからなる RQ が使用するメモリ領域を，単純化のために配列化したとしても，RQ のメモリ使用量は OS 全体で使用するメモリ量に比べて極めて小さいため，RQ を図 3 のように実装した．この RQ は最大タスク数を N とすると，$N+1$ Byte 分のメモリしか必要としない．

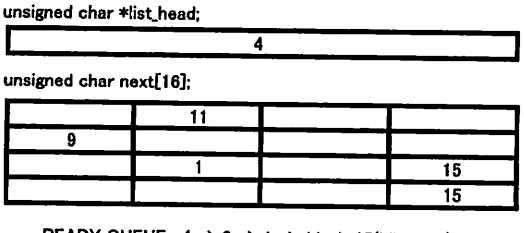

図 3 Ready Queue

3. 時間管理機構の構成

計算資源は有限であるため，リアルタイムシステムにおいては，負荷が上がると全てのタスクのリアルタイム性を保証することが出来なくなってしまう．一般的なリアルタイムシステムであれば，あるタスクをアボートすることによって一時的に負荷を下げ，他のタスクのリアルタイム性を確保することが可能である．しかしロボット制御においては，全てのタスクは依存関係にあるためアボート可能なタスクは存在せず，動的にリアルタイム性を確保する事は困難である．また，動的なアドミッションコントロールを行うためには，プロセッサに余分な性能が求められるため，組込みには向いていない手法であると言える．

一方，静的に負荷を下げるためには，タスクの周期を延ばす方法とタスクを破棄する手法の 2 通りが考えられる．まずタスクの周期を延ばす手法は，最終的にどんなタスクセットでもスケジュール可能となるため，柔軟性の面で優れているといえる．しかし，ロボット制御では実行周期によって使用するパラメータが変わってくる事があるため，OS 側が勝手にタスクの周期を変更することは望ましくない．タスクを破棄する手法では，OS 側からは各タスクの意味抽出まで行うことは困難であるため，破棄するタスクを決定するための指標は，優先度に限られる．しかし，優先度は基本的に時間制約の強さによってつけるものであり，優先度が低いからと言ってそのタスクが重要ではないとは限らない．優先度の他に重要度等のパラメータを導入することは不可能ではないが，メモリ使用量が増えてしまう．また，ロボット制御において，基本的に全てのタスクは依存関係にあり，重要ではないタスクというものは存在しない．

そこで本論文では，タスク生成時に，そのタスクが生成されることによって全てのタスクのデッドラインを保証可能かどうか静的に解析する．そして，何れかのタスクがデッドラインミスを起こす可能性があるようであれば，新たなタスクの生成を認めないという手法をとる．このアドミッションコントロールが正常に働きつづける限り，既に生成されているタスクについてはシステムのスタート後にデッドラインミスを起こさないことを保証できる．よって，アドミッションコントロールを行うことで，リアルタイム理論についてほとんど知らない人でも，簡単に周期タスクのリアルタイム性を保証できるようになる．ただし，システムに必要な全てのタスクがデッドラインミスを起こさないことを保証した上で，計算資源を有効に利用するためには，各タスクにおけるアルゴリズムのチューニングや，周期の調整を，アプリケーション開発者が手動で行う必要がある．

また，コード量削減のために，周期タスクの生成時しかアドミッションコントロールを行わないこととする．タイマ割込みサービスルーチン登録時にもアドミッションコントロールを行うことは可能であるが，どちらも用意するのは機能的に冗長であると言える．そして，周期タスク生成時のアドミッションコントロールの方が，演算量が比較的少なくて済むため，本研究で開発するリアルタイム OS では，組み込みという観点から，周期タスク生成時のアドミッションコントロールのみをサポー

トする．そのため，一度アドミッションコントロールをパスしたタスクでも，生成後に新たにタイマ割込みサービスルーチンが登録されると，デッドラインミスを起こす可能性が生じてしまう．よって，システム全体のリアルタイム性を確保するためには，周期タスクを生成する前に，全てのタイマ割込みサービスルーチンを登録する必要がある．

3.1 アドミッションコントロール

あるタスクセットがスケジュール可能であるかを判定する方法としては，全てのタスクの周期と最悪実行時間から詳細な解析を行う方法が，最も多くのタスクセットを保証できる可能性が高い．しかし，詳細な解析を行うルーチン自体に多くの計算資源を必要とするため，組み込みには向かないと言える．ここで，各スケジューリングアルゴリズムによって，資源利用率がある値以下であれば，どんなタスクセットにおいてもスケジュール可能であることを保証できる値というものがある．そこで本リアルタイム OS では，この上限値を用いてスケジュール可能性判定を行う．

RM アルゴリズムは，周期の短いものほど優先度が高くなる固定優先度アルゴリズムであり，スケジューラの計算量が少ない．しかし，理論的に任意のタスクセットについてスケジュール可能であることを保証できる資源利用率の上限は，タスク数が n のとき，以下の式で表わされることが証明されている [3]．この式によると，タスク数 n が多くなると，せいぜい 70 %弱までしか保証できないということになる．

$$U = n(2^{\frac{1}{n}} - 1)$$

実際の例として，以下のタスクセットを RM でスケジューリングしたときの様子を図 4 に示す．

- 周期タスク T_1: 実行時間 1，周期 5
- 周期タスク T_2: 実行時間 1，周期 5
- 周期タスク T_3: 実行時間 3，周期 7
- 周期タスク T_4: 実行時間 1，周期 7

このように RM では，資源利用率にはまだ余裕があるにもかかわらず，T_4 はデッドラインミスを起こしてしまう．一方，EDF アルゴリズムは，デッドラインの近いタスクほど優先度が高くなる動的優先度アルゴリズムであり，タスクがプリエンプティブであれば理論的にどんなタスクセットでもスケジュール可能であることが証明されている．よって，RM とは違い，計算資源を 100 %まで使い切ることが出来るという利点がある．ここで，先ほどのタスクセットを EDF を用いてスケジューリングすると，図 5 のように，T_4 もスケジュール可能となる．しかし動的優先度であるがゆえに，あるタイミングでは T_1，T_2 より T_3，T_4 が優先的に実行されてしまうため，T_1，T_2 の実行タイミングのジッタが大きくなってしまうという問題がある．その点，RM では T_1，T_2 のジッタが抑制されていることが分かる．このように，計算資源を有効に使うためには，実行タイミングのジッタを犠牲にしなくてはならないというトレードオフがある．また EDF は，スケジューラを呼び出すごとに優先度を更新しなくてはならないため，RM に比べスケジューラ自体のオーバーヘッドが大きくなってしまうという欠点もある．

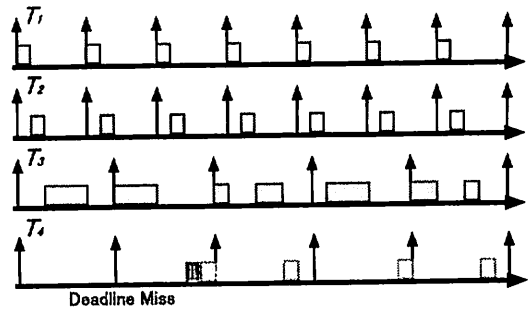

図 4 RM スケジューリング

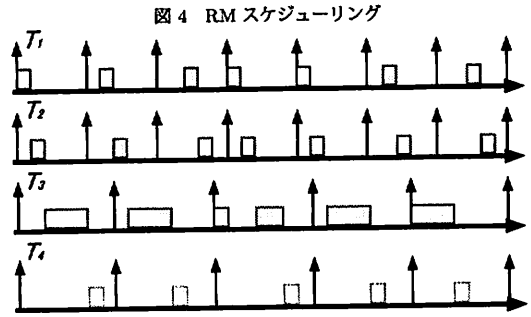

図 5 EDF スケジューリング

3.2 ジッタを許容しないタスクの分離

RM スケジューリングは，理論的に保証できる資源利用率の上限が低いだけであり，実際はスケジュール可能となるタスクセットは無数に存在する．そのため，リアルタイム性の保証がアプリケーション開発者に任されている既存のリアルタイム OS では，RM においてもタスクセットを工夫することで，特定のタスクのジッタを小さく維持しつつ全体の資源利用率を高めることが可能であった．しかし，本リアルタイム OS のように，OS 側でリアルタイム性を保証する場合，特定のタスクの実行タイミングのジッタを抑制しつつ，どんなタスクセットでも高い資源利用率を達成しなくてはならない．

そこで本研究では，タイマ割込みサービスルーチンをユーザが任意に使用可能とすることによって，ジッタを許容しないタスクをスケジューリングから分離する．ジッタを許容しないタスクをタイマ割込みサービスルーチンで実行したときの状況を図 6 に示す．この図から，高い資源利用率を維持したまま，特定のタスクのジッタを抑制していることが分かる．タイマ割込みサービスルーチンを用いることで，周期タスクのスケジューリングについて実行タイミングを考慮しなくて済むようになるため，EDF のようにジッタの生じやすい動的優先度アルゴリズムも適用することが可能となった．さらに，システムを実行している間は，それらのタイマ割込みをマスク不可とすることにより，タイマ割込みサービスルーチンはそれぞれのルーチンと他の割り込みだけにしか影響を受けなくなるため，従来の優先度を用いた制御に比べ，遥かにジッタの少ない周期実行が可能となる．

また，ロボット制御ならではの特性として，高度な制御を行うためには，アクチュエータ・センサの制御周期をできるだけ

短縮したいという要求がある．しかし，タスクの周期を短くすると，スケジューラの起動周期も短くしなければならないため，計算資源を占有する時間が長くなり，相対的にタスクの資源利用率を低下させる要因となっていた．ただし，ロボット制御において，短い周期を理想とするタスクとはセンサ・アクチュエータに直接関連するタスクである．よって，それらのタスクはすべてタイマ割込みルーチンで実行されることとなるため，周期タスクスケジューリングの際に無視することが出来るようになるため，スケジューラの起動周期を短くする必要が無くなり，相対的に計算資源を有効に使えるようになる．

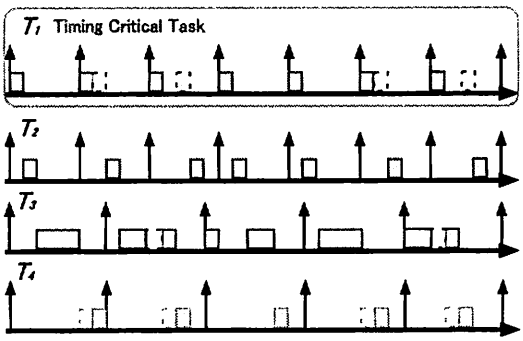

図 6　タイマ割込みサービスルーチンを用いた EDF

3.3　強制的なタイマ割込みの影響

タイマ割込みサービスルーチンの割り込みをマスク禁止にするということは，その部分だけ優先度アルゴリズムに従わずに強制的に実行されてしまうことになり，スケジューリングに大きな影響を与える．まず，RM においては，タイマ起動周期が周期タスクの周期より短ければ，高優先度のタスクとみなして解析することが可能である．しかし，タイマ起動周期が周期タスクより長い場合は，RM ではなくなってしまうため，理論的にスケジューラビリティを保証することが出来ない．そこで，RM においては，アドミッションコントロール時に，周期タスクの周期がタイマ割込み周期より長いときのみ生成を許可する．ただし，周期が長い上にジッタを許容しないタスクはロボット制御においてまず存在しないため，RM において強制的な割込みは大きな問題にはならないと考えられる．一方，EDF においては，強制的に割込まれることによって，優先度の逆転が起こり，周期タスクがデッドラインミスしてしまう可能性がある．そこで本論文では，優先度の入れ替わる可能性のある時間の最悪値を周期タスクの最悪実行時間に加え，資源利用率の計算を行う．割込みを考慮した最悪実行時間を E' とすると，それぞれの関係は以下の式に表される．

$$E'_n = E_n + \sum_{k=1}^{k=num} (e_k + 2c)$$

- E_n:n 番目の周期タスクの最悪実行時間
- e_k:k 番目のタイマ割込み処理の最悪実行時間
- num:タイマ割込みサービスルーチンの数
- c:コンテキストスイッチのオーバーヘッド

強制的な割込みが発生する環境においては，割込み処理ルーチンが長くなればなるほどアドミッションコントロールの際に，周期タスクの最悪実行時間を多く見積もらなくてはならなくなってしまうため，実際の資源利用率が低下してしまう可能性が高い．ただし，アクチュエータやセンサの制御を行うタスクについて，ジッタを許容しないのはアクチュエータに指令値を書き込むタイミングやセンサデータを読み出すタイミングだけであり，それ以外のフィードバックループ等の部分に関しては他のハードデッドラインタスクと同じように，デッドラインまでに結果が得られさえすれば周期内のどのタイミングで実行されても問題は無い．そこで，本研究で提案するリアルタイム OS 上では，図 1 に示すように，アクチュエータやセンサのデータを直接操作するタスクを，それぞれを制御するタスクとは別のタスクとして生成する．この方法によって，ロボット制御に限って言えば，割込みによる影響を軽減することが可能である．

また，タイマによって割り込まれたタスクは，割込み処理自体の実行時間に加え，2 回のコンテキストスイッチが発生する．コンテキストスイッチはプロセッサの動作周波数には依存せず，一定の命令数を必要とするため，動作周波数が比較的低い組み込みプロセッサにおいては，コンテキストスイッチによるオーバーヘッドが相対的に大きくなってしまう．そのため，タイマ割込みサービスルーチンの数が増えると，アプリケーション自体の資源利用率が RM で保証可能な上限値を下回ってしまう可能性もある．しかしながら近年のプロセッサは，割込み処理専用のコンテキストを持っていたり，複数のコンテキストを同時に保持したりすることが可能であったりして，割込み処理のコンテキストスイッチにかかるオーバーヘッドが無視できるほど小さくなっているものが多い．本論文でも，レジスタがバンク化されているプロセッサを用い，あるバンクを割込み処理専用とすることで，割込み時のコンテキストスイッチのオーバーヘッドを大幅に削減した．また，周期タスクとして実行される場合にもコンテキストスイッチは発生するため，むしろ割込みコンテキストを使える分，タイマ割込みサービスルーチンで実行した方が，全体的な資源利用率は高くなると考えられる．

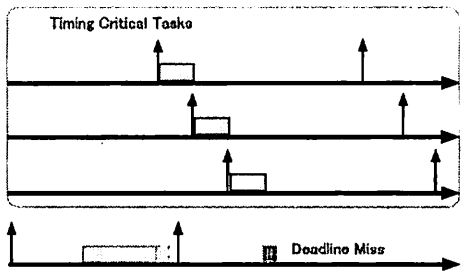

図 7　強制的な割込みの影響

4.　評　　価

EDF はアドミッションコントロールを行う上で，理論的には RM より資源利用率を高めることが可能だが，優先度を更新する処理や管理するパラメータの増加によって，スケジューラ

自体のコストが高くなってしまう．本章では，実装コストに注目して，EDF と RM の比較を行う．

本論文では，Virtex-5 LX50 の評価ボード上に，本研究室で開発中のコアである MULTI LIGHT(論文未発表) を用いて，本リアルタイム OS の性能を評価した．評価環境を表 1 にまとめる．これ以外にも，MULTI LIGHT はコア毎に 16 のレジスタバンクを持ち，高速なコンテキストスイッチが可能であるという特徴を持つ．

表 1　評価環境: MULTI LIGHT の構成

CPU	25 MHz × 2
	Register Bank × 16
Scratch Pad Memory	コア毎に 32 KB
I/O	Timer Unit × 8
	PWM Generator × 8
	UART Unit × 2
	Ethernet Counter × 1

4.1　スケジューラのオーバーヘッド

本リアルタイム OS において，スケジューラの明示的呼び出しのオーバーヘッドは，RQ は優先度リストであるため，スケジューリングアルゴリズムに関わらず常に一定である．そこで，スケジューラの暗黙的呼び出しのオーバーヘッドを比較する．一般的なシステムでは平均処理時間がスループットを上げる重要なパラメータとなるが，リアルタイムシステムではどんなに平均の実行時間が短くとも，最悪実行時間として見積もらなくてはならないため，出来るだけ最悪実行時間が短いほうがアドミッションコントロールを通過する可能性が高くなる．そこで，以下のタスクセットを実行したとき，割り込みコンテキストに入ってから処理を終えてユーザーコンテキストに戻るまでのサイクル数の最大値を図 8 に示す．その際，全ての状況が発生するまで十分な試行を行った．

- 20msec を基準として，20msec ずつ増加
  {20}, {20, 40}, ⋯ , {20, 40, ⋯ , 180, 200}

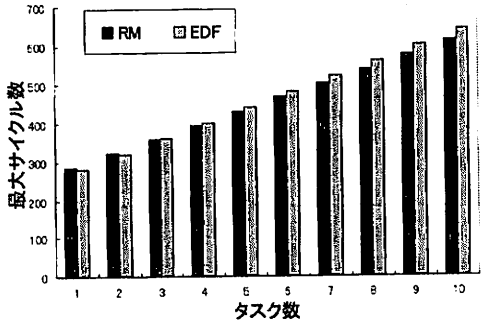

図 8　スケジューラの暗黙的呼び出しのオーバーヘッド

それぞれタスク数が増加するにつれて，最大サイクル数も線形に増加しているが，RM より EDF の方がサイクル数の増加の割合が大きい．これは，EDF は個々のタスクがリリースされる毎にデッドラインを計算し，優先度を更新しなくてはならないためである．そして，タスク数 10 のとき最大で 30 サイクルの差が生じている．本リアルタイム OS では，アクチュエータ・センサの制御を行うタスクをタイマ割込みサービスルーチンに任せることにより，短い周期実行を必要とするタスクを，周期タスクのスケジューリングから分離している．よって，一般的なロボット制御システムのタスク周期から，暗黙的呼び出しの起動周期をせいぜい 10msec として見積もると，EDF の導入によって生じる可能性のある 30 サイクル程度の余分なオーバーヘッドは，25MHz の低速なプロセッサにおいても，全体の処理時間の僅か 0.012 %に過ぎない．よって，EDF を導入しても，アプリケーションに割り当てることの出来る資源利用率の上限はほとんど低下しないと言える．

また，メモリの使用量についても表 2 にまとめる通り，432Byte の差があるが，全体の約 5 %にしか過ぎず，大きな問題にはならないと言える．

表 2　コードサイズ

アルゴリズム	コードサイズ
RM	9795 Byte
EDF	10227 Byte

5.　結　　論

本論文では，タスク生成時にアドミッションコントロールを行い，OS 側でタスクのデッドラインを保証する時間管理機構を提案した．OS による時間の管理は，ロボット制御のように多数の周期タスクからなるシステムに非常に有効である．また，タイマ割込みサービスルーチンと組み合わせることによって，特定のタスクのジッタを抑制することが可能となった．そして評価により，本論文で提案する時間管理機構は組み込みプロセッサにも充分適用可能であることを示した．

本時間管理機構を有効に利用するためには，各タスクの最悪実行時間を正確に見積もる必要がある．特に，タイマ割込みサービスルーチンで実行される処理については，アドミッションコントロール時のスケジュール可能性判定に大きな影響を与えるため，厳密に最悪実行時間を見積もる必要がある．

謝辞

本研究は，新エネルギー・産業技術総合開発機構 (NEDO) の支援による．

文　　献

[1] J.A. Stankovic, "Misconceptions about Real-Time Computing", *IEEE Computer*, Vol. 21, No. 10, pp. 10-19, 1988.

[2] E.S. Neo, K. Yokoi, S. Kajita and K. Tanie, "Whole-Body Motion Generation Integrating Operator's Intention and Robot's Autonomy in Controlling Humanoid Robots", *IEEE Transactions on Robotics*, Vol. 23, No. 4, 2007.

[3] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Envirnment", *Journal of ACM*, Vol.20, pp. 46-61, 1973.

[4] TRON Association, http://www.tron.org/.

[5] 石綿陽一，松井俊浩，國吉康，"高度な実時間処理機能を持つ Linux の開発"，日本ロボット学会学術講演会予稿集，Vol.16, pp.355-356, 1998.