計算機アーキテクチャ　99－10
（1993. 3. 12）

# スケジューリングポリシの動的変更に関する研究

飯田 浩二　矢向 高弘　菅原 智義　安西 祐一郎

慶應義塾大学理工学部

横浜市港北区日吉 3-14-1
慶應義塾大学理工学部 電気工学科
安西・天野研究室

あらまし　ロボットの制御等には、リアルタイム性を考慮したスケジューリングが適当であり、同時に環境の変化等によって起動されるタスクは迅速に実行されなければならない。スケジューラはそれらの要求を満たさねばならないが、実行されるタスクの種類等の条件により、どのようなスケジューリングポリシが適当であるかは一概にはいえない。そこで、パーソナルロボット用 OS $\mu$- PULSER に最短デッドライン優先方式、レート・モノトニック方式等、種々のリアルタイムスケジューラを実装し、それらのロボット制御に対する有効性を様々な条件下で比較・検討する。さらに、その結果を元にスケジューリングポリシの動的変更機構を実装し、常に有効なポリシを選択するようにする。

和文キーワード　リアルタイムスケジューリング、自律移動ロボット

# Dynamic exchange of scheduling policies for autonomous robots

Kouji IIDA, Takahiro YAKOH, Tomoyoshi SUGAWARA and Yuichiro ANZAI

Keio University

Anzai-Amano Laboratory
Faculty of Science and Technorogy,Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama

Abstract

Real-time schedulings are needed for controlling autonomous robots, and tasks that invoked by change of environment must be serviced quickly. Schedulers must achieve these requirements, but to say what scheduling policy is suitable for autonomous robots is not easy because there are various kinds of tasks executed by system. So in this paper, we implement many real-time scheduling policies, such as rate monotonic/earliest deadline first, on $\mu$-PULSER, an operating system for personal robots, and evaluate them how adaptive for controlling personal robots. Moreover, we describe a mechanism for a dynamic exchange of policies to achieve suitable scheduling usually.

英文 key words　Real-time Scheduling, Autonomous Mobile Robot

# スケジューリングポリシの動的変更に関する研究

飯田 浩二　　矢向 高弘　　菅原 智義　　安西 祐一郎*

慶應義塾大学理工学部

## 1 はじめに

近年、自律移動ロボットに関する研究が盛んに行われている。我々の研究室では自律移動ロボットの将来像の一つとして、パーソナルロボット [10] を対象とした研究を行っている。我々の想定しているパーソナルロボットは、一般の人達が家庭やオフィス内で仕事や趣味として使用するようなものであり、その用途として例えば、お手伝いロボットといったようなものを想定している。このようなパーソナルロボットが動作する環境ではロボットの動作を妨害するような事象がいつでも発生し得るため、パーソナルロボットは環境の変化を常に監視し、その変化に対して機敏にかつ敏感に反応できるリアクティブ性が必要である。

パーソナルロボットはモータやセンサ等の物理空間に対するデバイスを搭載しており、それらを制御しながら行動する。これらのモータ制御、センシング等の処理は周期的に実行されることが望ましく、それを制御するオペレーティングシステムはリアルタイム性を保証する処理を行える必要がある。

我々が開発しているパーソナルロボット用オペレーティングシステム$\mu$-PULSER[7] は、パーソナルロボットに対して要求されるリアクティブ性を実現することを１つの目標としており、そのために、高速な同期機構を提供している。また、マイクロカーネル化した構成であるため、スケジューラを容易に交換することが可能である。

リアルタイム制御のためにはタスクの時間的制約を考慮したリアルタイムスケジューリングが有効である。しかし、パーソナルロボットにはリアクティブ性も要求されるため、どのようなスケジューリングアルゴリズムが適当かは一概にはいえない。

本研究では、パーソナルロボットに要求されるリアルタイム制御を実現するために、種々のリアルタイムスケジューラを$\mu$-PULSER 上に実装し、様々なタスクセットの下でその特性を比較・検討する。さらに、常に適当なスケジューリングポリシを選択するためにスケジューリングポリシの動的変更機構に関して検討する。

以下、次節ではオペレーティングシステム$\mu$-PULSERについて述べるとともに、パーソナルロボット制御に必要と考えられるスケジューラについて検討する。また、3節でリアルタイムスケジューラの設計と実装について述べる。そして4節で実装したスケジューラの評価を行い、5節で現在実装中のポリシの動的変更について述べる。最後に結論と今後の課題を述べる。

## 2 $\mu$-PULSER とリアルタイムスケジューリング

本節では本研究のターゲットオペレーティングシステムであるパーソナルロボット用オペレーティングシステム$\mu$-PULSER について簡単に述べる。また、パーソナルロボット制御に必要とされるスケジューラについて検討する。

ここで、リアルタイム性とリアクティブ性を次のように定義する。

- **リアルタイム性**
  周期、デッドライン、最悪実行時間といった時間的制約を持ったタスクを実行できることであり、結果の正確さとともに時間的な正確さも要求されること
- **リアクティブ性**
  非周期的に生じ、デッドライン、最悪実行時間といった時間的制約を持たないタスクを可能な限り早く実行できること

### 2.1 パーソナルロボット用オペレーティングシステム$\mu$-PULSER

我々が開発した$\mu$-PULSER はマルチタスク・マルチスレッドモデルに基づいたパーソナルロボット用オペレーティングシステムである。その構成要素はマイクロカーネルと複数の管理スレッドである。マイクロカーネルは割り込みハンドラと特権命令を実行するためのシステムコールなどを含み、ほとんどのサービスは交換可能な管理スレッドが提供する。管理スレッドとしては、タスク・スレッド管理スレッド、通信用のポート管理スレッド等が

*Kouji IIDA, Takahiro YAKOH, Tomoyoshi SUGAWARA and Yuichiro ANZAI

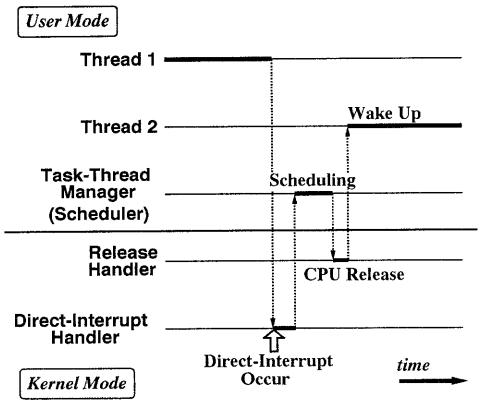


図 1: スケジューリングのタイミングチャート

ある。タスク・スレッド管理スレッドはタスクやスレッドの生成・削除・始動・停止等の処理を行い、スレッドのスケジューラもこの中に含まれている。

$\mu$-PULSER では各種割り込みによる同期機構とコンテキストスイッチ機構を統合したダイレクトインタラプト信号 (DI) と、それにより駆動される DI 駆動のスケジューラを提供している。DI は次のように動作する。ハードウェアやソフトウェアの割り込みにより DI が発生すると、ただちにスケジューリングが行われる。DI には識別子がついており、この識別子によって特定のスレッドが起動される (図 1)。このように DI によって起動されるスレッドを割り込みスレッドと呼ぶ。複数の割り込みスレッドが実行可能な場合は、それらをラウンドロビン方式で並行に実行し、割り込みスレッドがない時には通常のスレッド (継続スレッド) をラウンドロビン方式で実行する。

### 2.2 パーソナルロボット制御に必要なリアルタイムスケジューラ

前節で述べたように、$\mu$-PULSER はリアクティブ性を実現するための機構を提供している。さらにパーソナルロボット制御にはリアルタイム性が要求される。リアルタイム性はマニピューレータ制御や、一定周期でセンサをポーリングすることによるセンサ能動化 (Sensor Activation) 等に必要である。したがって、パーソナルロボット用オペレーティングシステムはリアルタイムスケジューラを提供する必要がある。

今日までに、数々のリアルタイムスケジューリングアルゴリズムが提案されているが、パーソナルロボット制御にはリアルタイム性と同時にリアクティブ性を実現するスケジューラが必要である。リアルタイムスケジューリングアルゴリズムはリアルタイム性を考慮してこそいるが、それらのスケジューリングアルゴリズムはリアクティブ性を実現するためのものではない。従って、パーソナルロボット制御にはどのようなリアルタイムスケジューリングアルゴリズムが適当であるかは一概にはいえない。どのようなスケジューリングアルゴリズムが適当であるか、種々のスケジューリングアルゴリズムについて実験的に評価する必要がある。

また外部状況や実行されるタスクの性質によっては、タスクを効率的に実行するために、スケジューリングポリシを動的に変更することが有効と考えられる。非周期的タスクが多く発生するような状況では、デッドラインの近いものを優先的に実行する最短デッドライン優先方式を選択し、割り込みが多く発生するような状況や、割り込みを優先して処理しなければならないような状況においては、そのためのスケジューリングポリシを選択し実行するというように、スケジューリングポリシを動的に変更することで、より適応性の高いスケジューリングポリシを選択する、などが考えられる。

## 3 リアルタイムスケジューラの設計と実装

本節では、$\mu$-PULSER へのリアルタイムスケジューラの設計と実装について述べる。スケジューラはポリシモジュールとメカニズムモジュールを分離し、相互インタフェースを定義することで、スケジューリングポリシの変更をポリシモジュールを変更することのみで実現する。

### 3.1 スケジューラの設計

本研究では、種々のスケジューリングポリシを実装してそれらのパーソナルロボット制御に対する適応性の評価をすることを一つの目標としている。従って、スケジューリングポリシの変更に伴うオペレーティングシステムに対する変更は可能な限り少なくし、また共用できる部分は可能な限り共用できることが、ポリシを交換して比較することを考慮すると好ましい。そこで本研究では、ARTS[8, 9] 等に見られるようなポリシ/メカニズムの分離の概念を取り入れた設計を行なった。またこれによりポリシを動的に変更する際にも、使用するポリシの変更が容易になるという利点がある。

$\mu$-PULSER は管理スレッドの一つであるタスク・スレッド管理スレッドが、タスクやスレッドの生成・削除・始動・停止といった処理を行ない、さらにスレッドのスケジューリングを行なっている。そこで、タスク・スレッド管理スレッドからスケジューリングポリシに依存した部分と依存しない部分、すなわちポリシとメカニズムを分離し、スケジューリングポリシの独立性を高め、スケジューリングポリシを交換可能にする。

```
struct thread_type {
  int   rttype;       /* スレッドの種類 */
  u_int arrivaltime;  /* スレッドの到着時間 */
  u_int starttime;    /* 実行開始可能時間 */
  u_int deadline;     /* デッドライン */
  u_int period;       /* 周期 */
  u_int calctime;     /* 最悪実行(計算)時間 */
}
```

図 2: スレッドの時間属性の記述

### 3.1.1 リアルタイムスレッドの導入

リアルタイムスレッドを扱うため、$\mu$-PULSER 上で実行されるスレッドを次の様に分類する。

**リアルタイムスレッド**: 時間制約を持ったスレッド

**割り込みスレッド**: DI によって起動されるスレッド

**継続スレッド**: 通常の状態で実行されるスレッド

リアルタイムスレッドはそのシステムへの到着の仕方により 2 種類に、さらにデッドラインにより 2 種類に分類する。したがって、以下のような 4 種類に分類される。

- **周期的スレッド**
  - **周期的ハードデッドラインスレッド**
    周期的に実行され、デッドラインを満足されなければならないスレッド
  - **周期的ソフトデッドラインスレッド**
    周期的に実行されるが、デッドラインを満足することを強要されないスレッド
- **非周期的スレッド**
  - **非周期的ハードデッドラインスレッド**
    非周期的に起動され、デッドラインを満足されなければならないスレッド
  - **非周期的ソフトデッドラインスレッド**
    非周期的に起動され、デッドラインを満足することを強要されないスレッド

リアルタイムスレッドを記述するためは、デッドラインや周期といった時間的属性をスレッドに持たせる必要がある。そこで、スレッドに関する時間的属性を図 2のように記述する。

`starttime` はリアルタイムスレッドがその実行を開始できる時刻、`period` は周期的スレッドの場合のスレッドの周期、`deadline` はスレッドのデッドラインを表す。周期的スレッドの場合、`deadline` は各周期ごとに次のように設定する。

```
arrivaltime = deadline
deadline = arrivaltime + period
```

リアルタイムスレッドは実行可能状態としてスケジューラに登録されているが、まだ `starttime` になってはいないので実行できないという状態をとり得る。

| | |
|---|---|
| `policy.init()` | スケジューリングポリシの初期化を行なう。またタスク・スレッド管理スレッド内にすでに存在している実行可能スレッドをポリシモジュールのキューに登録する。 |
| `policy.add(th)` | 実行可能スレッドをポリシに登録する。 |
| `policy.del(th)` | ポリシモジュールに登録された実行可能スレッドを削除する。 |
| `policy.choose()` | 実行するスレッドをスケジューリングポリシに従って選択する。 |
| `policy.end()` | ポリシモジュールの持つローカルなデータを破棄し、そのスケジューリングポリシを無効にする。 |

図 3: ポリシモジュールの定義

### 3.1.2 ポリシモジュールのインタフェース

各スケジューリングポリシモジュールは、タスク・スレッド管理スレッド内から呼び出されるモジュールの形をしている。スケジューラを交換可能とするために、各スケジューリングポリシを抽象化し、統一した呼び出しの形となるように実装する。それらを図 3のように定義する。

各スケジューリングポリシは、Rate Monotonic 方式であれば、`rm.choose()`、最短デッドライン優先方式では `dl.choose()` というように同一のインタフェースの形で記述する。

スケジューリングの効率を上げるため、ポリシモジュールはスケジューリングに必要な、タスク・スレッド管理スレッド内の次の情報を共有している。

- スレッドの属性
- システムの時刻 (`currentTime`)
- 実行中のスレッドの横取りの時刻 (`preemptTime`)

スレッドの属性はタスクスレッド管理スレッド内で保持するデータをポリシモジュールと共有するが、ポリシモジュールからは属性の変更は行わない。システムの時刻は、ポリシモジュールがスケジューリングのために現在のシステムの時刻を知るために使用する。実行するスレッドの横取りの時刻は、ポリシモジュールが低レベルタイマハンドラにより横取りまたはコンテキストスイッチを起こさせるために設定する。

スレッドの横取りや、コンテキストスイッチが起こるたびにタスク・スレッド管理スレッドは呼び出される。

### 3.1.3 スケジューリングポリシ

実装するスケジューリングポリシとして次のものを選択した。これらのスケジューリングポリシについては今日まで数々の研究成果が得られており、リアルタイムスケジューリングポリシとしては代表的なものであると考えられる。

- レート・モノトニック (Rate Monotonic) [2, 3, 5]
- Rate Monotonic + Sporadic Server [6]
- 最短デッドライン優先 (Earliest Deadline First) [1, 4]

これらのスケジューリングポリシは基本的にリアルタイムスレッドを割り込みスレッドより優先して実行するようにする。

Rate Monotonic 方式は周期的タスクを周期の短いものほど優先してスケジューリングする横取り可能なスケジューリング方式である。Sporadic Server 方式は、Rate Monotonic 方式を改良した方式であり、非周期的タスクサーバを用いて非周期的タスクの応答時間を向上させるスケジューリング方式である。Earliest Deadline First 方式は、デッドラインの近いタスクを優先してスケジューリングする横取り可能なスケジューリング方式である。

## 3.2 スケジューラの実装

前節で述べたような設計方針に基づき、スケジューラを実装した。本節では、スケジューラの実装の詳細について述べる。

### 3.2.1 実装環境

現在、$\mu$-PULSER は MC68EC030[1](25MHz) を CPU とする AVME-130-1 ボード (RAM 8MB、ROM 2MB、シリアルポート×3、パラレルポート×1、イーサポート×1) 上に実装されている。このボードは本研究室で開発中の VME バス結合型マルチプロセッサ・ロボットのメインボードである。

### 3.2.2 タスク・スレッド管理スレッドの変更

タスク・スレッド管理スレッドからは、図3で示したような方法で、ポリシモジュールを呼び出すようにした。

周期的スレッドはスレッドのデッドラインと実行開始時刻を各周期が終るたびに設定し直すようにした。また、1ms 単位でデッドライン等の時間属性を設定できるようにした。

### 3.2.3 各スケジューリングポリシの実装

各スケジューリングポリシごとにポリシモジュールを実装した。ここでは、一例として Rate Monotonic 方式の実装について述べる。

Rate Monotonic 方式では、ポリシモジュール内にローカルなキューとして以下のものを用意した。

[1]MC68030 互換であるが、MMU を搭載していない。

`ptTop_rm` 周期的スレッドを周期の小さいものから並べたキュー

`aptTop_rm` 非周期的スレッドをデッドラインの順に並べたキュー

`evTop_rm` 割り込みスレッドのキュー

`contTop_rm` 継続型スレッドのキュー

これらのキューに保持されるデータ構造は双方向リストの形をしており、タスク・スレッド管理スレッドで保持しているスレッド構造体を参照するためのポインタを持つ。このスレッド構造体へのポインタは、タスク・スレッド構造体から `rm.add(th)` の引数として渡されるようにした。スケジューリングに必要なスレッドの属性はこのポインタを用いて取得する。

`rm.add(th)` の呼び出しがあると、Rate Monotonic 方式ではスレッドの属性によって上記のようなキューに分類して、スレッドを保持する。周期的スレッドの場合は、周期の小さなものから順にキューに保持する。このためこのキューにはスレッドが優先順位順に並ぶことになる。Rate Monotonic 方式の場合、非周期的スレッドをスケジューリングする方法を提供していないが、非周期的スレッドの場合はハードデッドライン、ソフトデッドラインを考慮せずにデッドラインの順にスケジューリングするようにした。非周期的スレッドは実行可能な周期的スレッドが存在しない時に実行されるようにした。イベント型、継続型のスレッドはそれぞれのキューの先頭に入れ、次にイベント型または継続型のスレッドが実行可能になった時には最初にそのスレッドが選択されるようにした。

スレッドの横取りや、割り込みスレッドや継続スレッドのコンテキストスイッチを起こさせるために、`rm.choose()` はそのための時刻を `preemptTime` という大域変数にスレッドの選択時に設定する。

基本的にリアルタイムスレッドはそのスレッドの実行が完了する時刻に、また割り込みスレッドまたは継続スレッドは 10ms ごとのコンテキストスイッチをする時刻に `preemptTime` を設定する。もし選択したスレッドより優先順位の高いリアルタイムスレッドが存在し、そのスレッドの実行開始時刻が設定したコンテキストスイッチ時刻よりも早ければ、そのリアルタイムスレッドの実行開始時刻を横取りする時刻として `preemptTime` に設定する。

Sporadic Server 方式でも同様な方法で実装したが、非周期的タスクサーバとして周期的スレッドをスケジューラ内で疑似的に用意することで、Sporadic Server 方式を実装した。

Earliest Deadline Forst 方式では、周期的スレッドと非周期的ハードデッドラインスレッドを `edlTop` というキューでデッドライン順に保持するようにした。周期的ス

レッドのデッドラインは、周期の終りとしている。また非周期的ソフトデッドラインのスレッドを保持するキュー(aptTop_dl)、さらに割り込みスレッド(evTop_dl)、継続スレッドを保持するキュー(contTop_dl)を用意した。スレッドをedlTop、aptTop_dl、evTop_dl、contTop_dlの優先順位で選択するようにした。

# 4 評価

本節では実際に$\mu$-PULSER上に実装したリアルタイムスケジューラの実験的評価結果について述べる。

## 4.1 測定環境

測定は次のような方法により行った。時間の測定には、CPUボードに用意されている外部カウンタを使用し、これにより1$\mu$s単位で測定を行った。リアルタイムスレッドの実行時間には、あらかじめ外部カウンタを利用して最悪実行時間を測定した値を用いた。

## 4.2 評価の方針

実装したスケジューリングポリシのパーソナルロボット制御に対する適応性を評価するために、以下のような項目について測定を行った。

- スケジューラのオーバヘッドの検討
- DIの反応時間
- 非周期的タスクの反応時間

これらの測定により、パーソナルロボット制御用のOSとしてスケジューラに要求されるリアクティブ性とリアルタイム性を評価する。

## 4.3 スケジューラのオーバヘッド

スケジューリングポリシごとのオーバヘッドとして、policy.choose()の所要時間を測定した。周期的スレッドの数を変化させ、その時のpolicy.choose()に要した時間の1000回の平均値を測定した。この結果を図4に示す。ここで、RM BGはRate Monotonic方式を、RM SSはSporadic Server方式を、またDLは最短デッドライン方式での測定結果を示す。

policy.choose()にかかるオーバヘッドについては、今回の実装では最短デッドライン方式の場合が最もオーバヘッドが大きい。レートモノトニック方式とSporadic Server方式では、Sporadic Server方式の方が処理が複雑であるため、前者に比べオーバヘッドが数$\mu$sから数十$\mu$sほど大きくなっている。

## 4.4 DIに対する反応時間

$\mu$-PULSERにおいてDIは同期機構を実現するための機構であり、リアクティブ性に関しては非常に重要であるため、高速に処理される必要がある。ここではスケジューリングポリシごとにDIの反応時間を測定した。

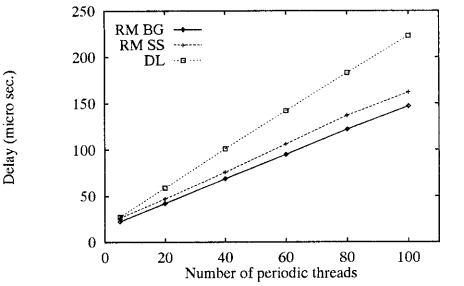


図4: 周期的スレッド数に対するpolicy.choose()のオーバヘッド

表1: 無負荷時のDIに対する反応時間

| スケジューリングポリシ | RM BG | RM SS | DL |
|---|---|---|---|
| DIのレスポンスタイム($\mu$s) | 268.0 | 221.0 | 249.0 |

まず、リアルタイムスレッドが存在しない状態でスケジューリングポリシごとにDIに対する反応時間を測定した。この結果を表1に示す。これはポリシごとのDIに対する最低反応時間を示す。

次に周期的スレッドの存在下でのDIに対する反応時間を次のように測定した。10個の周期的スレッドによりいくつかCPUの負荷を変え、タイマ割り込みによりダイレクトインタラプトを起こし、そのダイレクトインタラプト発生時からそのDIによって割り込みスレッドが実行されるまでの時間を測定した。この結果を図5に示す。図よりリアルタイムスレッドの増加とともに、反応時間が増大する傾向にあり、Rate Monotonic方式で周期的スレッドの負荷が約89%の時は、平均の反応時間が30ms近くかかっていることがわかる。

反応時間はどのスケジューリングポリシもCPU利用率が50%を越えたあたりから極端に悪くなった。これはどのスケジューリングポリシも、実行可能なリアルタイムスレッドが存在しない時にのみ割り込みスレッドを実行するようにスケジューリングするため、負荷が増えるに従い、割り込みスレッドが実行可能になる時間が減っていくためである。

低負荷時にはどのスケジューリングポリシでもDIに対する反応時間はそれほど増えず、同期機構も高速に動作することができるが、高負荷時には、DIに対する反応時間が大きくなるため、同期機構の高速性が失われてしまい、リアクティブ性を確保することは難しくなる。

したがって、高負荷時にリアクティブ性を確保するた

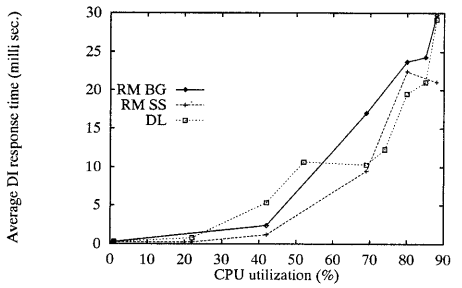


図 5: DI に対する反応時間

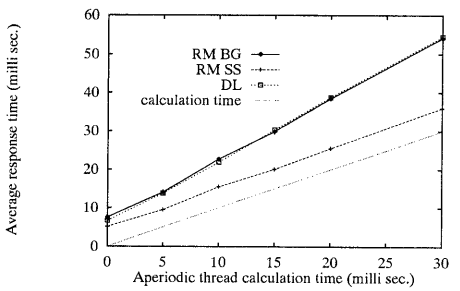


図 6: 非周期的スレッドの実行時間と平均応答時間の関係 (周期的スレッドによる負荷 42%の時)

めには割り込みスレッドを優先的にスケジューリングするべきであろう。

### 4.5 非周期的スレッドの応答時間の比較

スケジューリングポリシの比較として、ある周期的スレッドを実行している状態における非周期的ソフトデッドラインスレッドの応答時間を測定した。

非周期的スレッドの実行時間を変化させ、そのスレッドの実行終了までの時間を測定した。測定時は 10 個の周期的スレッドを用意し、それらのスレッドの実行時間を適当に設定することで、システムの負荷 42%、88%の 2 種類の状態を例として測定した。非周期的スレッドはソフトデッドラインであるとし、デッドラインは起動時から 100ms 後に設定した。図 6、7のそれぞれにおいて、calculation time で示された線は、その非周期的スレッドが実行を完了するまでに最低限必要な時間を示している。図 6で、Sporadic Server 方式では非周期的スレッドの遅れは 5ms 程度であり、また図 7でも他の 2 方式にの遅れに対して 25%程度の遅れで実行が完了している。このように非周期的スレッドのレスポンスタイムは Sporadic Server 方式が非常に良い結果を示している。

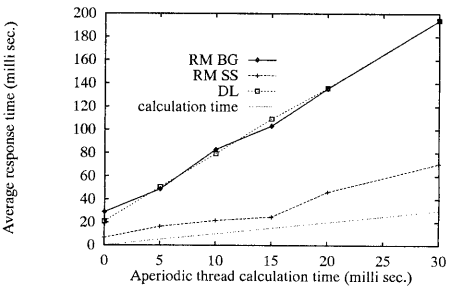


図 7: 非周期的スレッドの実行時間と平均応答時間の関係 (周期的スレッドによる負荷 88%の時)

## 5 スケジューリングポリシの動的変更

前節の評価より、ソフトデッドラインの非周期的スレッドに関しては、Sporadic Server 方式が適当であると考えられるが、非周期的ハードデッドラインスレッドが多く存在する状況では、むしろ Earliest Deadline First 方式が有効であることも考えられる。またリアルタイム性よりもリアクティブ性を必要とする必要のある危機回避等のためには、リアクティブ性を重視した、つまり割り込みスレッドを優先して実行できるようなスケジューリングポリシを選択するべきである。これまで実装したポリシでは割り込みスレッドを効率的に処理することが可能ではなかった。したがって、それぞれの状況にとって最適なスケジューリングポリシを動的に選択することが望ましい。そこで本節では現在実装中のスケジューリングポリシの動的変更機構について説明する。

スケジューリングポリシを変更することは、基本的には、それまでのポリシを `old_policy`、新しいポリシを `new_policy` とすると

1. 新しく使用するポリシの初期化をする (`new_policy.init()`)
2. それまでのポリシの使用を中止する (`old_policy.end()`)
3. 使用するポリシの変更をする (`schedPolicy` に新しいポリシ番号を代入)

とすることで行う。この時注意しなければならないこととして、ポリシを変更する際に、リアルタイムスレッドがデッドラインを満足できなくなることは許されないということがある。それはスケジューリングポリシを変更

することによって、ポリシモジュールの初期化、終了といったことに対するオーバヘッドが生じたり、ポリシの変更によりスレッドの実行順序が変わり、それまでデッドラインを満足できるようにスケジューリングされていたスレッドがデッドラインを満足できなくなる可能性があるからである。

このようなことを避けるために、ポリシを変更する時には、

- 実行可能なリアルタイムスレッドがない
- リアルタイムスレッドが実行可能になるまでにポリシの初期化が終了する

という条件をつけることで、ポリシの終了、初期化に伴うオーバヘッドを吸収する。

「実行可能なリアルタイムスレッドがない」とは、実行可能時刻を過ぎているが、まだ実行されていないリアルタイムスレッドが存在しないという状態であり、ポリシモジュールが `policy.choose()` によりリアルタイムスレッド以外のスレッドを実行するスレッドとして選択するような状態であるということができる。リアルタイムスレッドが実行可能になるまでにポリシの初期化を終了させるには、スレッド数に対するポリシの初期化 `policy.init()` に要する最悪の場合の時間を予測し、`policy.choose()` によって設定された `preemptTime` と現在時刻との差をとることでポリシの初期化を行うために十分な時間が存在するかを判定し、十分な時間がある場合に行うようにする。

## 6 結論と今後の課題

本論文では、パーソナルロボット用 OS $\mu$-PULSER へ種々のリアルタイムスケジューラを実装し、それらのパーソナルロボット制御に対する適応性を、リアルタイム性とリアクティブ性という観点から評価した。またスケジューリングポリシの動的変更についても述べた。

リアルタイム性とリアクティブ性を兼ね備えたスケジューリングは、これまでのポリシでは実現することができなかった。リアルタイム性とリアクティブ性を兼ね備えたスケジューリングポリシを実現するため、今後はリアルタイム性を保証しつつ、割り込みスレッドを優先して実行できるようなスケジューラを研究、実装していく必要がある。また、常に最適なスケジューリングを行えるようにポリシを動的に変更する機構については、今後も十分に検討していく必要がある。

## 参考文献


[1] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp. 1261–1269, 1989.

[2] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. of IEEE REAL-TIME SYSTEMS SYMPOSIUM*, pp. 166–171. IEEE, 1989.

[3] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, Vol. 20, No. 1, pp. 46–61, 1973.

[4] K. Schwan and H. Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, Vol. 18, No. 8, pp. 736–748, 1992.

[5] L. Sha and S. S. Sathaye. Architectural support for real-time computing using generalized rate monotonic theory. 計測と制御, Vol. 31, No. 7, pp. 744–756, 1992.

[6] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *The Journal of Real-Time Systems*, Vol. 1, No. 1, pp. 27–60, 1989.

[7] 菅原 智義, 飯田 浩二, 秋庭 朋宏, 紺田 和宣, 矢向 高弘, 安西 祐一郎. パーソナルロボットのための敏感で柔軟な OS $\mu$-PULSER の設計と実装. 日本ソフトウェア科学会第 *9* 回大会, pp. 253–256, 1992.

[8] H. Tokuda, M. Kotera, and C. W. Mercer. An integrated time-driven scheduler for the ARTS kernel. In *Proc. of 8th IEEE Phenix Conf. on Computers and Cummunications*, pp. 486–496. IEEE, March 1989.

[9] H. Tokuda and C. W. Mercer. ARTS: A distributed real-time kernel. *ACM Operating Systems Review*, Vol. 23, No. 3, pp. 29–53, 1989.

[10] 山崎 信行, 安西 祐一郎. パーソナルロボットのためのアーキテクチャの提案. ロボティクス・メカトロニクス講演会 *'92* 講演論文集, volume A, pp. 51–56, 1992.