マイクロカーネル Lavender における 2レベルスケジューラの構成方式

毛利公一[†] 大久保英嗣[††]

本論文では，マイクロカーネル Lavender における2レベルスケジューラの構成方式について述べる．本スケジューラは，ポリシとメカニズムの分離の概念に基づいて構成されている．すなわち，ポリシをユーザ定義可能なローダブルカーネルモジュールとし，メカニズムをカーネル内モジュールとして実現している．さらに，ポリシモジュールは，各々のスケジューリングアルゴリズムそのものを実現するスケジューラモジュールと，異なるポリシによって生じるスケジューリングの競合を解決する調整モジュールの2つのモジュールに分類される．以上の構成方式を採用することによって，スケジューラ自体のカスタマイズ性を向上させ，複数のスケジューリングアルゴリズムの共存が可能となる．本論文では，2レベルスケジューラの実装と性能評価について述べ，その有効性について議論する．

A Construction Method of Two-level Scheduler in Lavender Micro Kernel

KOICHI MOURI[†] and EIJI OKUBO[††]

In this paper, a construction method of two-level scheduler in Lavender micro kernel is described. This scheduler is constructed based on the concept of policy/mechanism separation. Namely, the policy is constructed as a user-specific loadable kernel module. On the other hand, the mechanism is constructed as a intra-kernel module. Furthermore, the policy module is classified into two modules: the scheduler module in which each scheduling algorithm is implemented, and the coordinator module which solves scheduling conflicts caused by different policies. By means of this construction method for two-level scheduler, the scheduler itself becomes to be customizable and several scheduling algorithms can be independently executed each other. In this paper, an implementation and performance evaluation of two-level scheduler are described, and discussions of its effectiveness are presented.

1. はじめに

マイクロプロセッサの高速化とネットワーク技術の進展により，多様なアプリケーションが開発されている．これにともない，そのためのプラットフォームであるオペレーティングシステム（以下，OS と記す）にも多様な機能が求められ，その拡張性が重要となってきている．このため，現在，拡張可能な OS の研究がさかんに行われている[1)~3)]．OS の機能を拡張可能とするための機構としては，Mach[4)]の外部ページャや SPIN[5)]の spindle などがあげられる．このような拡張可能な OS では，特に，アプリケーションの性能を考慮したスケジューリングが重要となる．しかし，スケジューリングに関しては，拡張性・カスタマイズ性からの検討およびそれに基づいた実際のシステムの実現が十分に行われていないのが現状である．

アプリケーション固有のスケジューリング機構を実現する手法としては以下のものがある．

(1) 複数のスケジューリング方式をあらかじめシステム側で用意し，アプリケーションに適した方式を選択する手法．この手法は，Spring[6)]や RT-Mach[7)]における各種リアルタイムスケジューリングおよび同期方式の提供や，Choices[8)]における各種スケジューリングクラスの提供に代表される．

(2) ポリシとメカニズムの分離の概念により，スケジューリングに必要となる汎用的・共通的機能をメカニズムとしてシステムが提供し，アプリ

† 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†† 立命館大学理工学部情報学科
Department of Computer Science, Faculty of Science and Engineering, Ritsumeikan University

ケーション固有の処理をポリシとして簡単に実現するためのインタフェースを提供する手法．この手法は，USS（Universal Scheduling System）[9),10)]に代表される．

(3) デバイスドライバの中にスケジューラを実現する手法．たとえば，周期的なクロック割込みを処理するドライバの中にスケジューリングのための処理を実装する．この手法は，ターゲットに組み込まれるリアルタイムOSと，システム開発用のUNIXやWindows NTとの共存を可能とする場合に使用されることが多い．特徴的なものとしてはWindows NTのHAL（Hardware Abstraction Layer）を改造し，Windows NTとリアルタイムOSへの割込みを振り分ける方式を採用しているRTX[11)]がある．

(4) オブジェクト指向におけるリフレクション機能[12)]を使用して，スケジューリングに関するメソッド呼び出しをフックし，独自にアプリケーションに適したスケジューリングアルゴリズムを実現する手法．この手法は，Apertos[13)]などに代表される．また，コルーチンによるユーザレベルスケジューラやユーザレベルスレッドライブラリも同様の方式である．この手法には，cthread[14)]やpthread[15)]がある．

以上のような手法が考えられるが，我々は，ユーザカスタマイズ性を重視し，基本的に(2)の手法を採用することとした．それは，次のような理由による．まず，(1)の手法のように，複数のスケジューリング方式をあらかじめシステム側で用意しても，アプリケーションが必要とするスケジューリング方式と必ずしも一致しない場合がある．また，(3)の手法では，デバイスドライバの内部にスケジューラを実現するため，ユーザは，オペレーティングシステムの内部構造を理解し，スケジューリング機構をすべて構築する必要がある．さらに，(4)の手法では，複数のスケジューリングアルゴリズムを共存させるための機構が提供されていないため，アプリケーション固有のスケジューリング機構を1台のプロセッサ上で複数動作させることができない．したがって，結果的にアプリケーションの必要とするプロセッサ割当てを行うことができない．また，ユーザレベルスケジューリング機構では，システム全体のスケジューリングアルゴリズムを変更することができないことも問題である．

(2)の代表であるUSSでは，優先度関数，調停規則，決定モードの3つによって，任意のスケジューリングアルゴリズムを実現することが可能である．しかし，これは，複数のスケジューリングアルゴリズムを共存させることは考慮されていない．本論文で提案する手法は，その点も考慮し，さらに以下のことを目的としている．

(1) カーネルレベルスケジューリング機構をユーザがカスタマイズ可能とすること．

(2) 複数のスケジューリングアルゴリズムを共存可能にすること．

(3) スレッド単位でスケジューリングアルゴリズムを指定できること．

(4) 各スケジューラは他のスケジューラが動作していることを意識しなくてもよいこと．

我々は，これらの目的を実現するために，ポリシとメカニズムの分離を基本とした2レベルスケジューラをマイクロカーネルLavender[16)]上に実装した．以下，本論文では，2章でスケジューラの全体構成，3章でスケジューラの処理方式について述べる．さらに，4章では，スケジューラの性能評価について述べ，その有効性について議論する．

2. スケジューラの全体構成

Lavenderにおけるスケジューラの全体構成を**図1**に示す．スケジューラは，メカニズムモジュールとポリシモジュールの2階層から構成される．

メカニズムモジュールは，ポリシ（すなわち，スケジューリングアルゴリズム）を実現するために必要となる共通的な機能を提供する．共通的な機能としては，スケジューラを呼び出すためのインタフェース，ポリシに従ってスレッドへプロセッサを割り当てる機能，ポリシモジュールの管理がある．そのためのインタフェースとして，ディスパッチャインタフェース，中継インタフェース，キュー操作インタフェースが提供されている．これらのインタフェースは，各々の機能を提供するために優先度別キュー，スケジューラインタフェーステーブル，調整インデックス，調整インタフェーステーブルを用いる．メカニズムモジュールは，システムにただ1つであり，カーネル内に配置されている．

ポリシモジュールは，プロセッサをどのスレッドにいつどれだけ割り当てるかを決定するものである．ポリシモジュールは，ユーザが定義可能なモジュールであり，各々のポリシごとに作成され，システムに複数配置することが可能である．また，スケジューリングのオーバヘッドを軽減するために，ローダブルカーネルモジュールとしてカーネルのアドレス空間に配置される．ポリシモジュールは，動的な組込み，削除，交

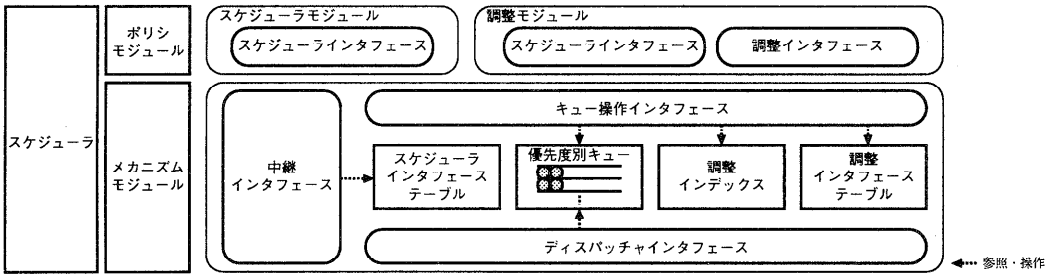

図1 スケジューラの全体構成
Fig. 1 Overview of scheduler.

換が可能となっている．ただし，動的な組込みなどを行う場合の，安全性の確認やスケジューリング情報の継承はユーザの責任において行う必要がある．

ポリシモジュールは，役割の違いによって，さらにスケジューラモジュールと調整モジュールに分けられる．スケジューラモジュールは，スレッドのスケジューリングを行う．そのためのインタフェースとして，スケジューラインタフェースを持つ．調整モジュールは，スケジューラモジュールの操作や，スケジューラモジュール間の調整を行う．そのためのインタフェースとして，スケジューラインタフェースと調整インタフェースを持つ．

3. スケジューラの処理方式

Lavender におけるスケジューラは，単一プロセッサ構成のマシンを想定しており，プロセス・スレッドモデルを対象としている．プロセスは，メモリ保護の単位であり，スレッドに対してメモリ資源を提供する．スレッドは，レジスタセットとスタックを持った実行実体であり，プロセスが提供するメモリ資源内で動作する．スレッドはプロセスに属し，1つのプロセス内に複数のスレッドが属していてもよい．Lavender のスケジューラでは，スレッドをスケジューリングの単位としている．また，Lavender のカーネルの内部は，システムコールやタイマなどによって割込み管理部に制御が移り，そこからカーネル内部のモジュールを呼び出す実行形態となっている．

以下，スケジューラにおける処理の流れと各構成要素の処理方式について述べる．

3.1 スケジューラにおける処理の流れ

スケジューラは，タイマ割込みによって割込み管理部から呼び出される．また，スレッドが生成または破棄されたときやスレッドの状態が変更されたときにプロセス管理部，同期機構，排他制御機構，シグナル機構から呼び出される．スケジューラが呼び出された後

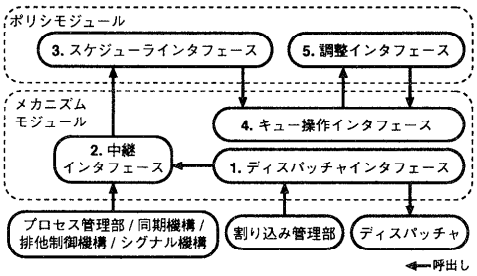

図2 2レベルスケジューラにおける処理の流れ
Fig. 2 Processing flow in two-level scheduler.

の処理の流れを以下に示す（図2参照）．

(1) ディスパッチャインタフェースは，スケジューリング行うために中継インタフェースを呼び出す．
(2) 中継インタフェースは，スケジューリングを行うポリシモジュールのスケジューラインタフェース☆を呼び出す．
(3) スケジューラインタフェースは，スケジューリングを行う．その結果を優先度別キューに反映するためにキュー操作インタフェースを呼び出す．
(4) キュー操作インタフェースは，調整を行う場合，調整インタフェースを呼び出す．調整を行わない場合は，優先度別キューの操作を行い，呼び出し元に戻る．
(5) 調整インタフェースは，調整を行い，その結果を優先度別キューに反映するためにキュー操作インタフェースを呼び出す．

3.2 メカニズムモジュール

メカニズムモジュール内の優先度別キューは，優先度ごとに分けられたレディ状態のスレッドをつなぐためのキューである．これは，ポリシモジュールにより決定されたスケジューリング結果を格納するためのも

☆ ここでのスケジューラインタフェースは，スケジューラモジュールと調整モジュール内にあるインタフェースの両方を意味している．

のである．各キュー要素には，以下の3つの値が格納される．

- スレッド識別子またはポリシモジュール識別子
- スレッドまたはスケジューリングの開始時刻
- 開始時刻からのプロセッサ割当て期間

単一プロセッサ上で実現可能なすべてのスケジューリングアルゴリズムにおいて，そのスケジューリング結果はこれらの3つの値で表すことができる．すなわち，これらは，種々のスケジューリングアルゴリズムに共通の要素である．ただし，スケジューリングを行うためには，スケジューラに制御を移行させる必要がある．Lavenderでは，スケジューラの実行形態はスレッドによるものではないが，制御を移行するためにスレッドと同様の方法でプロセッサの割当てのタイミングを指定する．このタイミングの指定の方法は，スレッド識別子の代わりにポリシモジュール識別子を指定することによって行われる．

スケジューラインタフェーステーブルは，システム内で動作しているポリシモジュール内のスケジューラインタフェースのアドレスを保持している．調整インデックスは，ポリシモジュール間の調整を行うか否かを管理するためのものである．また，本インデックスは，調整を行う場合は，どのスケジューラモジュールが，どの調整モジュールによって調整されるかの対応を管理する．調整インタフェーステーブルでは，調整モジュール内の調整インタフェースのアドレスを保持している．キュー操作インタフェースが調整モジュールを呼び出す際に用いられる．

以下，メカニズムモジュール内の3つのインタフェースについて述べる．

3.2.1 ディスパッチャインタフェース

ディスパッチャインタフェースは，優先度別キューの内容に従ってスレッドまたはポリシモジュールにプロセッサを割り当てる．キューの優先度は，キュー要素の開始時刻よりも優先して参照される．すなわち，低い優先度のキューにあるキュー要素は，開始時刻が過ぎている場合でも，高い優先度のキュー要素がプロセッサを放棄するまではプロセッサを割り当てられない．プロセッサを割り当てる処理は，以下に示す手順で実行される（**図3** 参照）．

(1) 直前に実行していたスレッドが実行可能か否かを調べる．実行可能でない場合は，最高の優先度のキュー要素を選択する．

(2) 実行可能な場合は，当該スレッドに割り当てた期間が残っているか否かを調べる．残っていない場合は，最高の優先度のキュー要素を選択する．

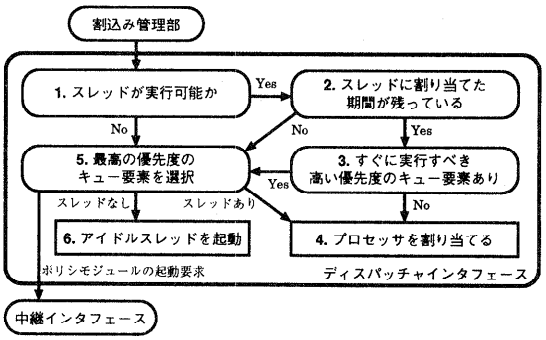

図3 ディスパッチャインタフェース内の処理の流れ
Fig. 3 Processing flow in dispatcher interface.

(3) 残り期間がある場合は，さらに高い優先度のキュー要素で実行開始時刻に到達したものがあるか否かを調べる．あれば，最高の優先度のキュー要素を選択する．

(4) 上記のいずれにも該当しない場合，すなわち，残り期間があり，ほかに優先度の高いキュー要素がない場合，引き続き直前に実行していたスレッドにプロセッサを割り当てる．

(5) 実行開始時刻に到達しているキュー要素のうち，最高の優先度のものを選択する．キュー要素がスレッドの場合は，そのスレッドに対してプロセッサを割り当てる．ポリシモジュールの場合は，中継インタフェースを経由してポリシモジュールを呼び出す．

(6) 選択すべきキュー要素がない場合はアイドルスレッドを起動する．

この方式によって，スケジューリングアルゴリズムごとに優先度を割り当てるようなスケジューラを容易に実現できる．特に，スレッド自体がプロセッサを放棄する場合のように，ポリシモジュールが予期できない事象が発生する場合に有効である．たとえば，優先度の高いスレッドがプロセッサを放棄した場合に，優先度の低いスレッドにプロセッサを割り当てる処理や，優先度の低いスレッドを実行中に優先度の高いスレッドがプロセッサを横取りするための処理が該当する．このような場合，ポリシモジュールを呼び出すことなく，メカニズムモジュール内でスレッドを切り替えることによって，オーバヘッドを軽減することが可能となる．

3.2.2 中継インタフェース

中継インタフェースは，スケジューラインタフェーステーブルの管理を行う．また，メカニズムモジュールからスケジューラインタフェースを呼び出すための処理を行う．この処理は，スケジューラインタフェー

ステーブルを参照し，スケジューラインタフェースを呼び出すことによって行われる．

3.2.3 キュー操作インタフェース

キュー操作インタフェースは，ポリシモジュールからの優先度別キューに対する操作要求を受け付ける．ポリシモジュールは，このインタフェースを用いてスケジューリング結果を格納する．ただし，キュー操作を要求したスケジューラモジュール間の調整を行う必要がある場合は，調整インデックス内で定義されている調整モジュールに対して要求が転送される．

キュー操作インタフェースで提供される機能と，それによってポリシモジュールが可能となる処理を以下に示す．

- キュー内のすべての要素の削除
  - すでにスケジューリングした内容を取り消す．
- キューへの要素の追加
  - スケジューリング結果を格納する．
- キュー全体の情報の取得・更新
  - 現在のスケジューリング状況を取得する．
  - スケジューリング結果を一括登録する．

キュー操作インタフェースを提供することによって，次のような利点が得られる．すなわち，ポリシモジュールは，スケジューリングの結果をキュー操作インタフェースに通知するだけでよい．これによって，ポリシモジュールからカーネルの内部構造を隠蔽することができ，ユーザがカーネルの内部を意識せずポリシモジュールを構築することが可能となる．

3.3 ポリシモジュール

ポリシモジュールは，ユーザが定義可能なモジュールであり，ユーザが処理方式を自由に定義することができる．ただし，メカニズムモジュールから呼び出される各インタフェースの呼び出し方式は規定されている．ポリシモジュールは，メカニズムモジュールから呼び出されるとき，要求パケットを受け取る．要求パケットは，処理要求コードとパラメータから構成される．処理要求コードは，ポリシモジュールの各機能に対応している．パラメータは，各機能を実行するために必要となるものである．この要求パケットを用いることで，ユーザは処理要求コードを追加することによってポリシモジュールの機能を拡張することが可能となる．

また，ポリシモジュールは，スケジューラインタフェースが呼び出されるタイミングを自分自身で指定することが可能となっている．これは，Lavenderにおけるスケジューリング機構の特徴的な機能である．これによって，ポリシモジュールが，能動的にスケジューリングを行うことができる．

以下，スケジューラモジュールと調整モジュールの処理方式について述べる．

3.3.1 スケジューラモジュール

スケジューラインタフェースで提供される機能は，ユーザがカスタマイズすることが可能である．ただし，以下に示す機能については，カーネル自体が必要とする機能であるため，必須としている．

- 初期化
- スケジューリング
- スレッドの追加・削除
- スレッドの状態変更

初期化は，システムの起動時に1度だけ呼び出される．スケジューラモジュール内で利用する各種情報の初期化を行う．スケジューリングは，スケジューラモジュール自体がスケジューリングされたときに呼び出される．ユーザは，この部分にスケジューリングアルゴリズムを実装すればよい．また，キュー操作インタフェースに対する操作もこの部分に実装される．スレッドの追加・削除，スレッドの状態変更は，それぞれの事象が発生した際に呼び出される．スケジューラモジュール内のキューの操作を行ったり，スケジューリングを行う．

スケジューラモジュールは，自分自身でスケジューリングのタイミングを決定することが可能であることから，たとえば，ユーザは，スケジューラに対して以下のことを指定することが可能となる．

- 周期的にスケジューリングを行う．
- プロセッサがアイドル状態になった場合にスケジューリングを行う．
- スレッドがタイムスライスを使い切るたびにスケジューリングを行う．

3.3.2 調整モジュール

調整モジュールは，スケジューラインタフェースと調整インタフェースを持つ．調整モジュールのスケジューラインタフェースの構成は，スケジューラモジュールのものと同様である．ただし，調整のためのスケジューリングを必要としない場合は，スケジューラインタフェースの機能を調整モジュール内に実装する必要はない．

調整インタフェースは，調整の対象となるスケジューラモジュールがキュー操作インタフェースを呼び出すことによって制御が渡される．このとき，調整インタフェースは，スケジューラモジュールからキュー操作インタフェースに対して渡された処理要求を要求パケットとして受け取る．したがって，調整インタフェースで

は，メカニズムモジュールのキュー操作インタフェースと同様の機能を提供する．調整インタフェースは，要求パケットを参照し，スケジューラモジュール間の調整を行う．調整の結果は，スケジューラモジュールと同様に，キュー操作インタフェース対する操作によって反映される．

各スケジューラモジュールは，互いの存在を意識していない場合，他のスケジューラがいつスレッドにプロセッサを割り当てようとしているかを知ることができない．複数のスケジューラが，同じ時刻に異なるスレッドに対してプロセッサを割り当てるスケジューリング結果を生成する場合，スケジューラ間で競合が発生する．調整モジュールを用いることで，この競合を解決することができる．

また，調整モジュールは，スケジューラモジュールと同様，自分自身でスケジューリングのタイミングを決定することが可能である．さらに，他のスケジューラモジュールからの，スレッドやスケジューラモジュールへのプロセッサ割当て要求を調整することで，それらへのプロセッサ割当てのタイミングを調整することが可能である．これによって，ユーザは，以下に示すような調整を行うことが可能となる．

- 複数のスケジューリングアルゴリズムの共存
- スケジューリング対象のスレッドの入替え
- スケジューリングアルゴリズムの変更
- 非同期イベントへの対応

4. 性能評価

Lavenderの2レベルスケジューラの評価を行うために，ラウンドロビンとレートモノトニック[17]のためのスケジューラモジュールを構築した．ラウンドロビンスケジューリングのためのスケジューラモジュールは，初期化，スケジューリング，スレッドの追加・削除・状態変更の処理を行うための機能を持つ．本モジュールは，C言語で約130行のプログラムとなっている．レートモノトニックスケジューリングは，周期的な処理を対象とした横取り可能なリアルタイムスケジューリングアルゴリズムである．各々の処理の周期と処理時間をパラメータとして，スケジューリング可能性を判定する．レートモノトニックのためのスケジューラモジュールでは，スケジューリングを行うときに起動時刻に達しているスレッドがあるか否かを調べ，周期スレッドを起動する．また，スレッドを追加するときにスケジュール可能性を調べる．本モジュールは，C言語で約260行のプログラムとなっている．

現在のところ，作成したスケジューラや，それを用いるアプリケーションが少ないため，スケジューラモジュールの構築の容易さに関する評価は十分に行えていない．これに関しては，様々なアプリケーションに適用して評価を行う予定である．ただし，ポリシとメカニズムを分離し，本論文で提案した方式を採用することによって，スケジューラの変更を行う際にカーネルの再構築や再起動などが不要となり，スケジューラの組込みが容易になると考えている．

表1 平均ターンアラウンド時間
Table 1 Average turnaround time.

0レベル (ms)	1レベル (ms)	2レベル (ms)
133,145.3	133,167.1	133,242.2

以下の評価は，PC/AT互換機（Pentium 166 MHz）上で動作するLavender上で行った．また，タイムスライスは1 msとしている．

4.1 スケジューリングオーバヘッド

2レベルスケジューラのオーバヘッドを評価するために，ラウンドロビンスケジューリングを用いて，(1) スケジューラを呼び出さない場合（0レベル），(2) 1レベルスケジューラの場合，(3) 2レベルスケジューラの場合の3つについて，整数型変数のインクリメントを100億回繰り返す処理にかかる時間を測定した．測定では，スレッド数を1としている．

実行結果を**表1**に示す．表は，上記3つのレベルにおけるスレッドの平均ターンアラウンド時間を示している．この結果から，スケジューリングを1回行うためにかかる時間は，1レベルが(133,167.1−133,145.3)/133,167より1.64 μsとなる．2レベルは，(133,242.2−133,145.3)/133,242より7.27 μsであることが分かる．

1レベルと2レベルでは，5.63 μsの差がある．この差は以下のものによる．

- 優先度別キューの参照
- スケジューラモジュールの呼び出し
- キュー操作インタフェースの呼び出しとキュー操作
- 上記の呼び出しの際の要求パケットの生成と解析

次に，レートモノトニックスケジューリングを用いて，以下に示す条件の下で各スレッドの起動間隔を求めた．ただし，実験の都合上，スケジューリング可能性の判定を行わないようにしている．

- スレッドは2個とする．
- スレッドの処理時間は1,331.5 ms，周期は処理時間の倍の2,663 msとする．
- スレッド1を起動してから，1,331.5 ms後にスレッド2を起動する．

この結果，これらのスレッドは周期が2,689.4 msとなった．したがって，レートモノトニックのスケジューラモジュールのオーバヘッドは，(2,689.4−2,663)/2,689より，呼び出し1回あたり9.82 μsとなる．ラウンドロビンと比較してこの値が大きいのは，スレッドの状態変更が発生した際に，キューの内容をソートする処理が必要となるからである．さらに，スケジューリングを行うたびに起動すべきスレッドを検索することも必要となる．

上記の結果を基に，さらに以下の実験を行った．

- スレッド数を3とする．
- スレッドは調和スレッドとする．調和スレッドは，すべてのスレッドの周期が，最も短い周期の2^n倍となっているスレッドの集合である．
- 処理時間は，全スレッドのプロセッサ利用率の合計が100%となるようにしている．
- スレッドの処理時間，周期，最初の起動時刻は以下のとおりである．

  スレッド 1 ：1,331.5 ms，2,663 ms，0 ms

  スレッド 2 ：1,331.5 ms，5,326 ms，1,331.5 ms

  スレッド 3 ：2,663 ms，10,652 ms，3,994.5 ms

この結果，スレッド1とスレッド2は周期を守ることができた．しかし，スレッド3は周期をオーバーランし，104.6 msだけ周期がずれる．

そこで，スレッド3の処理内容を変更し，104.6 msのオーバヘッドの分だけ処理時間を短くした．その結果，上記のスレッド1からスレッド3のすべてが周期を守って起動することができた．これは，最大周期のスレッドを，オーバヘッドを吸収するためのスレッドとし，オーバヘッド分だけ処理時間を短くすることによって，他のスレッドの周期を保証することが可能となることを意味する．

4.2 調整モジュールの評価

4.2.1 スケジューリングアルゴリズムの共存

調整モジュールによって，ユーザがスケジューラ間の調整を行うことが可能であること，および複数のスケジューリングアルゴリズムが共存可能か検証するために調整モジュールの構築を行った．

調整の対象として，ラウンドロビンとレートモノトニックのスケジューラモジュールを利用する．調整モジュールは，レートモノトニックスケジューリングを基本として，リアルタイムスレッドを高い優先度でスケジューリングする．しかも，空き時間にラウンドロビンで非リアルタイムスレッドのスケジューリングを行う．調整モジュールの大まかな処理内容を図4に示す．

```
void SchedulerInterface(RequestPacket_t *packet)
{
}

void CoordinatorInterface(RequestPacket_t *packet)
{
  switch(packet->RequestCode) {
  case THREAD_ADD:          /* スレッドの追加 */
    switch(packet->Sender) {
    case RATEMONOTONIC:     /* レートモノトニック */
      スレッドの優先度を0に設定;
      break;
    case ROUND_ROBIN:       /* ラウンドロビン */
      スレッドの優先度を1に設定;
      break;
    }
    メカニズムモジュールへスレッドの追加を依頼;
    break;
  }
}
```

図4 レートモノトニックとラウンドロビンのための調整モジュール
Fig. 4 Coordinator module for rate monotonic and round-robin.

この調整モジュールは，スケジューラインタフェース内部での処理を必要としないため，その実体は空の関数としている．調整インタフェースでは，スケジューラモジュールによって用いられるスレッドの追加の機能を実装している．すなわち，スレッドの追加時に調整を行う手法を採用している．各スケジューラモジュールからキュー操作インタフェースを経由してスレッド追加の要求を受け，優先度の変更による調整を行う．優先度は，レートモノトニックをラウンドロビンよりも高くする．調整を行った後，キュー操作インタフェースを用いてスレッドを優先度別キューに追加する．これにより，リアルタイムスレッドにプロセッサを割り当てている間は，非リアルタイムスレッドにはプロセッサが割り当てられない．本モジュールは，C言語で約40行のプログラムとなっている．

動作させるスレッドは以下のとおりである．

- リアルタイムスレッド数を2とする．
- リアルタイムスレッドは調和スレッドであり，処理時間，周期，最初の起動時刻は以下のとおりである．

  スレッド 1 ：1,331.5 ms，2,663 ms，0 ms

  スレッド 2 ：1,331.5 ms，5,326 ms，1,331.5 ms
- 非リアルタイムスレッド数を1とする．

実行の結果，リアルタイムスレッドは周期を守って起動された．さらに，プロセッサが空いたときに非リアルタイムスレッドが正しくスケジューリングされた．以上のことから，調整モジュールによって，異なるスケジューリングアルゴリズムを共存させることが可能であることが分かる．

また，上記の調整モジュールによる調整のオーバヘッ

```
void SchedulerInterface(RequestPacket_t *packet)
{
  switch(packet->RequestCode) {
  case INITIALIZE:          /* 初期化 */
    調整インタフェース用キューを作成;
    ActiveScheduler=Aのポリシモジュール識別子;
    入れ替えの時刻を指定し自分自身を追加依頼;
    break;

  case SCHEDULING:          /* スケジューリング */
    スケジューラモジュールAを優先度別キューから削除;
    スケジューラモジュールBを優先度別キューへ追加;
    ActiveScheduler=Bのポリシモジュール識別子;
  }
}

void CoordinatorInterface(RequestPacket_t *packet)
{
  switch(packet->RequestCode) {
  case THREAD_ADD:          /* スレッドの追加 */
    if(packet->Sender == ActiveScheduler)
      メカニズムモジュールへスレッドの追加を依頼;
    break;
  }
}
```

図5 スケジューリング対象のスレッド入替えのための調整モジュール
Fig. 5 Coordinator module for mode change.

ドは，2.85 μs であった．計測には，Pentium のクロックをカウントするレジスタを利用した．このオーバヘッドは，以下によるものである．

- 調整モジュールの呼び出しと調整
- キュー操作インタフェースの呼び出しとキュー操作

4.2.2 スケジューリング対象のスレッドの入替え

調整モジュールによって，モード変更，すなわちユーザがスケジューリング対象のスレッドの入替えが可能であることを検証するための調整モジュールの構築を行った．

調整の対象として，ラウンドロビンのスケジューラモジュール A と B の2つを利用する．それぞれのスケジューラモジュールには異なるスレッドを登録する．調整モジュールは，指定した時刻にスケジューラモジュールを切り替えることによって，結果的にスケジューリング対象のスレッドを入れ替える．調整モジュールの大まかな処理内容を**図5**に示す．

この調整モジュールは，スケジューラインタフェース内に初期化とスケジューリングの機能を持つ．初期化の機能では，調整用キューを生成し，スケジューラモジュール A を動作中のスケジューラモジュールとする．また，スレッド入替えのためのスケジューリングのタイミングを指定する．スケジューリングの機能では，優先度別キューを操作し，A を終了させて B を起動する．これによってスレッドを入れ替える．また，動作中のスケジューラモジュールを B とする．スケジューリングの機能は，初期化の際に指定した時刻に呼び出される．調整インタフェースでは，スレッドの追加の機能を実装している．スレッド追加の要求が来ると，動作中のスケジューラモジュールからのものであれば優先度別キューに反映する．本モジュールは，C 言語で約 90 行のプログラムとなっている．

実行の結果，調整モジュールが指定した時刻においてスケジューリング対象のスレッドが入れ替えられ，スケジューラモジュール B に登録されたスレッドが正しくスケジューリングされた．以上のことから，調整モジュールによって，スケジューリング対象のスレッドを入れ替えることが可能であることが分かる．

上記の調整モジュールにおける，スケジューリング対象のスレッドを切り替えるためのオーバヘッドは，10.2 μs であった．計測には，Pentium のレジスタを利用した．このオーバヘッドは，以下によるものである．

- 調整モジュールの呼び出し
- スケジューラモジュールの削除と追加
- スケジューラモジュールの呼び出し
- スレッドの登録

5. おわりに

本論文では，Lavender における2レベルスケジューラと性能評価について述べた．スケジューラを，メカニズムモジュール，スケジューラモジュール，調整モジュールの3つのモジュールで構成することによって，ユーザがカスタマイズすることが可能であることを示した．本方式によって，ユーザは，複数のスケジューリングアルゴリズムを共存させたり，調整することが可能となる．また，スレッド単位でスケジューリングアルゴリズムを指定することも可能となる．

性能評価においては，リアルタイムスケジューリングアルゴリズムとしてレートモノトニックを実際に動作させた．それによって，リアルタイムスレッドが周期的に動作可能であることを示した．また，2レベルスケジューラのオーバヘッドを計測し，オーバヘッドを吸収するスレッドを動作させることで，その他のスレッドの周期を守ることができることも示した．さらに，調整モジュールの評価として，スケジューリングアルゴリズムが共存可能であることと，スケジューリング対象のスレッドの入替えが可能であることを示した．

今後の課題として，本論文で言及しなかった同期機構とリアルタイム同期プロトコルに関する実装と評価があげられる．また，複数のプロセッサや分散環境への対応があげられる．

参考文献

1) Bershad, B.N., Savage, S., Pardyak, P., Sirer,

E.G., Fiuczynski, M.E., Becker, D., Chambers, C. and Eggers, S.: Extensibility, Safety and Performance in the SPIN Operating System, *Proc. 15th Symposium on Operating Systems Principles*, pp.267–284 (1995).
2) Cheung, W.H. and Loong, A.H.S.: Exploring Issues of Operating Systems Structuring: From Microkernel to Extensible Systems, *Operating Systems Review*, Vol.29, No.4, pp.4–16 (1995).
3) Small, C. and Seltzer, M.: A Comparison of OS Extension Technologies, *Proc. USENIX 1996 annual technical conference*, pp.41–54 (1996).
4) Accetta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: A New Kernel Foundation for Unix Development, *Proc. 1986 Summer USENIX Conference*, pp.93–113 (1986).
5) Bershad, B.N., Chambers, C., Eggers, S., Maeda, C., McNamee, D., Pardyak, P., Savage, S. and Sirer, E.G.: SPIN – An Extensible Microkernel for Application-specific Operating System Service, Technical Report, UW-CSE-94-03-03, Department of Computer Science and Engineering, University of Washington (1994).
6) Stankovic, J.A. and Ramamritham, K.: The Spring Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol.8, No.3, pp.62–72 (1991).
7) Tokuda, H., Nakajima, T. and P., R.: Real-Time Mach: Towards a Predictable Real-Time System, *Proc. Usenix Mach Workshop*, pp.1–10 (1996).
8) Leyens, D.E.: A Choices Implementation of the Universal Scheduling Systems, Technical Report, TTR 89-15, Department of Computer Science, University of Illinois at Urbana-Champaign (1989).
9) Ruschitzka, M. and Fabry, R.: A Unifying Approach to Scheduling, *Comm. ACM*, Vol.20, No.7, pp.469–477 (1977).
10) 市岡秀俊，安東一真，大久保英嗣，津田孝夫：オブジェクト指向オペレーティングシステム Ozone におけるプロセス管理方式，情報処理学会論文誌，Vol.32, No.11, pp.1401–1411 (1991).
11) Greban, R.: NT in Real Time, *BYTE*, Vol.10, chapter Features, McGraw-Hill (1996).
12) 渡部卓雄：チュートリアルリフレクション，コンピュータソフトウェア，Vol.11, No.3, pp.5–14 (1994).
13) Yokote, Y.: The Apertos Reflective Operating System: The Concept and Its Implementation, *Proc. OOPSLA'92*, pp.414–434 (1992).
14) Cooper, E.C. and Draves, R.P.: C Threads, Technical Report, CMU-CS-88-154, School of Computer Science, Carnegie Mellon University (1988).
15) IEEE: [IEEE/ANSI Std 1003.1, 1996 Edition] Information Technology – Portable Operating System Interface (POSIX) – Part 1: System Application: Program Interface (API) [C Language] (1996).
16) 毛利公一，山田博士，斎藤彰一，中村素典，大久保英嗣：マイクロカーネル Lavender における階層化インタフェース，情報処理学会研究報告 95-OS-70 (1995).
17) Liu, C.L. and Layland, J.W.: Scheduling Algorithm for multiprogramming in a Hard Real Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).

(平成 10 年 12 月 1 日受付)
(平成 11 年 4 月 1 日採録)

毛利 公一（学生会員）

昭和 47 年生．平成 8 年立命館大学大学院理工学研究科修士課程情報システム学専攻修了．同年同大学大学院理工学研究科博士課程後期課程総合理工学専攻に入学し，現在に至る．オペレーティングシステム，分散システム，実時間システム，インターネット等に興味を持つ．

大久保英嗣（正会員）

昭和 26 年生．昭和 52 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同年（株）日立製作所ソフトウエア工場入所．主として FORTRAN コンパイラの開発に従事．昭和 54 年京都大学工学部情報工学科助手．昭和 60 年同講師．昭和 62 年同助教授．平成 3 年立命館大学理工学部情報学科教授となり，現在に至る．工学博士．オペレーティングシステム，データベースシステム，分散システム，実時間システム等の研究に従事．著書に「オペレーティングシステムの基礎」（単著，サイエンス社），「情報工学実験」，「情報処理入門」（いずれも共著，オーム社）がある．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，ACM，IEEE-CS 各会員．