オペレーティング・システム 48－3
（1990. 9. 7）

Muse オペレーティングシステムにおける リアルタイム機能について

藤波順久、横手靖彦、寺岡文男、光澤 敦[1]、所 真理雄[2]
(株) ソニーコンピュータサイエンス研究所

今日の計算機システムでは、リアルタイム機能が必須になりつつある。本稿では Muse オペレーティングシステムの、階層化され方針と機構が分離されたスケジューラにおける、リアルタイム機能の実現に関する考察から、リアルタイムスケジューリングでは方針よりはそれが満たす性質が重要であることを明らかにする。さらに、Muse において柔軟性のあるリアルタイム機能の位置付けを考察する。その結果 Muse では、各階層のスケジューラにリアルタイムの方針を持たせるのは不適当で、一つの “ベーススケジューラ” と複数の “意図した方針” を用いている。実装における注意点も述べる。

Real-time Scheduling Facilities for the Muse Operating System

Nobuhisa Fujinami, Yasuhiko Yokote, Fumio Teraoka, Atsushi Mitsuzawa, and Mario Tokoro
Sony Computer Science Laboratory Inc.

Real-time facilities are indispensable to computer systems in these days. This paper reveils that importance for the real-time scheduling is the properties not the policies, through the study of the realization of the real-time facilities in the hierarchical schedulers in the Muse operating system. Thus putting a real-time policy in each hierarchy proves to be inadequate. We adopt one “base scheduler” and multiple “intended policies” to realize the flexible real-time facilities. Suggestion for the implementation is also included.

[1]慶應義塾大学理工学部
[2](株) ソニーコンピュータサイエンス研究所および慶應義塾大学理工学部

1 はじめに

ラップトップ計算機などの可搬型計算機の登場は、新しい計算機の利用法を要請している。例えば、ボイスメイル、FAX データ送受信、アラームクロック、等である。これらのアプリケーションで扱う情報は、利用者に密着して即時に処理する必要がある。例えば、ボイスメイルにおいて、他のタスクによってサンプリングタスクの実行が妨害されることは、再生される音声の質を低下させることになる。そのため、リアルタイム機能が重要になる。

Muse オペレーティングシステム [Yokote 89a] [Yokote 89b] は、可搬型ホストも含めた超大規模分散システムの構築を目指している。ここでいう可搬型ホストとは、システムに接続された状態でも、システムから切断された状態でも同じ環境で利用可能である。Muse はオブジェクト指向計算モデルに基づいて構築されており、その特徴は open かつ self-advancing なシステムである。この実現のため、我々は、オブジェクトとメタオブジェクトを分離し、メタ階層を導入している。オブジェクトはそのメタオブジェクトによってサポートされている。例えば、オブジェクトのローカル記憶はメモリメタオブジェクトによってサポートされる。オブジェクトは複数のメタオブジェクトによってサポートされ、それらはメタスペースを構成している。Muse ではメタオブジェクトもオブジェクトとして定義されるので、メタオブジェクトをサポートするためのメタスペースが存在する。このようにメタスペースは階層構造を成す。

このメタ階層はスケジューラの構成にも当てはまる。オブジェクトをスケジュールしているのはメタスペース内のスケジューラメタオブジェクトであるが、メタスペースには複数のメタオブジェクトが存在しているので、そのためのスケジューラが必要である。本稿では、このような Muse の階層化スケジューラにおけるリアルタイム機能の実現に関する考察から、リアルタイムスケジューリングではアルゴリズムよりはそれが満たす性質が重要であることを明らかにする。さらに、Muse において柔軟性のあるリアルタイム機能の位置付けを考察する。

以下本稿では、まず第 2章で代表的なリアルタイムスケジューリング法を紹介する。次に第3章で Muse オペレーティングシステムの構成を概観する。第4章で Muse におけるリアルタイム機能を位置付け、どのように実現していくかを述べる。第5章では Muse のリアルタイム機能を実装する上で注意するべき点を指摘する。最後に第6章で本稿のまとめ、今後の予定を述べる。

2 リアルタイムスケジューリング

リアルタイムシステムで重要なことは、各タスクがデッドラインを満たすこと— ある決められた時刻までにタスクの処理が終わることである。通常、実行したいタスクセットに対してすべてのデッドラインを満たすスケジューリングができる (スケジュール可能と言う) ようにシステムを設計する。スケジュール可能かどうかは、スケジューリングに使うアルゴリズムによって違ってくる。

シングルプロセッサのリアルタイムスケジューリングアルゴリズムの代表的なものには、rate monotonic スケジューリングアルゴリズム (RM と略す) と deadline driven スケジューリングアルゴリズム (DL と略す) がある [Liu 73]。これらのアルゴリズムでは、次のような性質のタスクを仮定している。

- タスクはそれぞれ等間隔で起動される。
- タスクのデッドラインはそのタスクが次に起動される時刻である。
- タスクの一回に必要な計算時間は決まっている[3]。
- タスクは任意の時点で中断可能である。
- 各タスクは独立である。

ここで、このようなタスクが m 個あるとし、その周期を $T_1, T_2, \cdots, T_m$、計算時間を $C_1, C_2, \cdots, C_m$ とすると、その CPU 利用率 U は

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

である。

優先順位が固定のとき、つまり、複数のタスクの実行要求があるときにどれを優先するかの順位が時刻によらずにあらかじめ決まっているようなアル

[3] 上限がわかっていればよく、一定である必要はない。

ゴリズムの中では、RM が最適なスケジューリングを与える。その方法は、タスクの周期の短い順に高い優先順位を割り当てていくというものであり、どのような周期や計算時間のタスクでも、CPU 利用率が

$$U \leq m(2^{\frac{1}{m}} - 1) \quad m\text{はタスク数}$$

ならスケジュール可能である。現実にはもっと高い利用率までスケジュール可能な場合が多い [Lehoczky 87]。

優先順位を動的に割り当てる場合、DL が最適なアルゴリズムとなる。その方法は、デッドラインが近いタスクを優先するというもので、$U = 1$ までスケジュール可能である。つまり、DL は RM より優れたアルゴリズムである。例えば、$m = 2$ で、$T_1 = 12, C_1 = 6, T_2 = 16, C_2 = 7$ の場合、

$$U = 0.9375 > 2(2^{\frac{1}{2}} - 1) \simeq 0.83$$

であり、RM ではスケジューリングに失敗する (図 1) が、DL ではうまくいく (図 2)。しかしながら、RM はハードウェア割り込みコントローラで実現しやすいなどの利点があり、実用上は重要である。

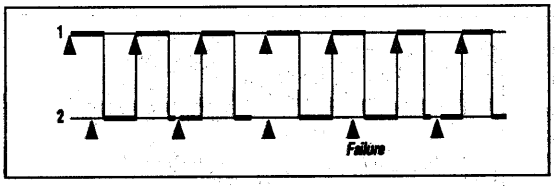

図 1: RM によるスケジューリング

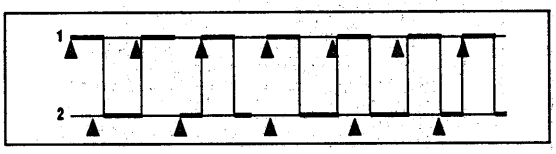

図 2: DL によるスケジューリング
▲はタスクの起動時刻かつデッドライン

3 Muse オペレーティングシステムの概要

本章では、Muse オペレーティングシステムの概要を述べる。Muse の全体像、特にメタ階層に基づく Muse の構造を説明することは、次章からの議論を進める上で必要である。ここでは、オブジェクトとメタオブジェクトの分離、およびその階層化を中心に述べる。より詳しい議論は [Yokote 90] を参照のこと。

Muse の目標の一つは、*ultra large-scale, open, self-advancing, and distributed* 環境を構築することである。この環境では、サービスの種類、交信パス、必要資源、等は動的に変化している。すなわち、これらはオブジェクトの実行に先だって不確定の要素である。さらに、新たなサービスがシステムに取り込まれたり、古いサービスがシステムから取り除かれたりしている。システムは拡張しながら進化している。

このような環境で重要なことの一つは透過性の制御である。例えば、超大規模分散システムにおいて、交信遅延は避けられないものであるため、何らかの手法を用いてモデル化する必要がある。しかし、あるレベルにおいては、交信の際の遅延を意識するのは望ましくない。Muse におけるメタ階層はこの制御を可能にする。

3.1 オブジェクトとメタオブジェクトの分離

Muse におけるオブジェクトは並行オブジェクト [Yonezawa 87] の考えを基にしている。ここでは、一つのオブジェクトは、プライベート記憶、メソッド、仮想プロセッサの 3 つの要素からなる。プライベート記憶はオブジェクトの状態を保持するローカル記憶である。メソッドはプライベート記憶に対するアクセス権を有する。仮想プロセッサはメソッドの解釈実行を行なう。

Muse では仮想プロセッサに意味を与えるもの、仮想プロセッサをシミュレートするものとしてメタオブジェクトを導入した。図 3 にその概念図を示す。図中では、楕円はオブジェクトを表し、グラデーションのかかった楕円はメタスペースを表す。メタオブジェクトはメタスペース内に複数存在する。オブジェクトの実行はメタオブジェクトによってサポートされる。すなわち、メタオブジェクトはオブジェクトのための専用の仮想機械、あるいは最適化されたオペレーティングシステムとみなされる。

例えば、プライベート記憶を管理するのはメモリメタオブジェクトの仕事である。プライベート記憶

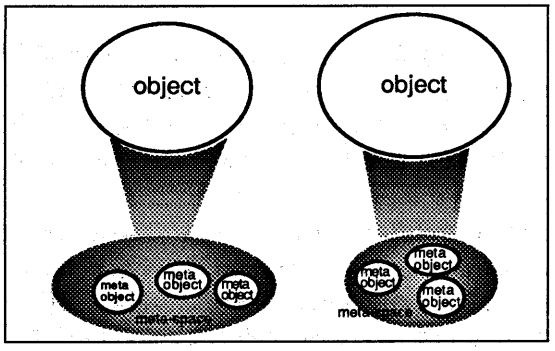

図 3: オブジェクトとメタオブジェクトの分離

で生じたページフォルトはメモリメタオブジェクトが解決する。また、オブジェクトのスケジューリングを行なうのはスケジューラメタオブジェクトである。これに関しては次章以降に詳しく述べられる。

Muse では、オブジェクトとメタオブジェクトの間にリフレクティブな関係を導入している。例えば、オブジェクトの実行コンテキストはメタオブジェクト内に表現される。従って、オブジェクトの実行コンテキストを操作することは、オブジェクトの実行に影響を与えることになる。

3.2 メタ階層

メタスペース内のメタオブジェクトもまたオブジェクトとして定義される。従って、それをサポートするためのメタオブジェクトが必要になる。図 4 に Muse におけるメタ階層を示す。メタスペース内には複数のメタオブジェクトが存在し、それらはメタスペース間で共有可能である。

3.3 実現例

今まで述べたオブジェクトのアーキテクチャは現在次のように実現されている。リフレクティブ機能を実現するために MuseCore とリフレクタを導入している。MuseCore は次のような機能を持つ。

- オブジェクトとメタオブジェクトの関係を保持する。
- オブジェクトとメタオブジェクト間の制御の移動を管理する。

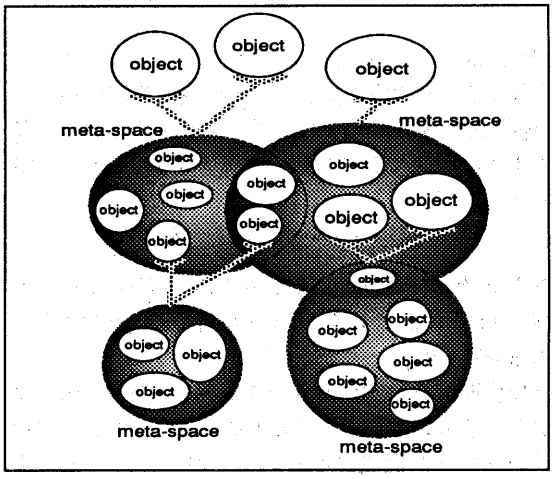

図 4: メタ階層

- 外部事象をメッセージに変えて適当なメタオブジェクトに送る。

一方、オブジェクトとメタオブジェクト間の計算の遷移は必ずリフレクタを介す。すなわち、オブジェクトはリフレクタを介してメタオブジェクトとやりとりが可能になる。図 5は Muse のオブジェクト構成を示したものである。図中で、影つきの楕円で表

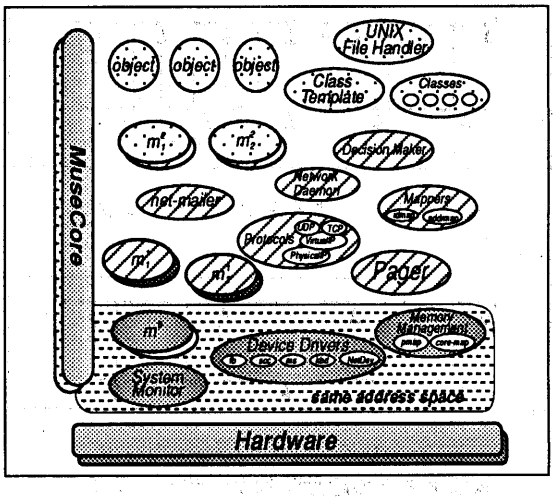

図 5: Muse におけるオブジェクト構成

されているのがリフレクタであり[4]、MuseCore はすべてのオブジェクトから利用される。

[4]以下本稿では、リフレクタは m_m^n で表し、これは n メタ階層の m 番目のリフレクタであることを表している。ここでは、m^0 を通してサポートされるメタ空間をメタメタ空間、m_m^1 を

4 階層化スケジューラとリアルタイム機能

ここでは、Muse オペレーティングシステムの階層化スケジューラにリアルタイム機能を持たせる際の問題を議論する。

4.1 Muse オペレーティングシステムのスケジューラ

Muse オペレーティングシステムにおいては、各メタスペースにスケジューラが存在し、そのメタスペースが管理しているオブジェクトのスケジューリングを行う。そのスケジューラ自身も、さらにそのメタスペースにあるスケジューラによってスケジューリングされている。こうして、CPU に一つあるメタメタ空間のスケジューラまで遡る。つまりスケジューラは階層化されることになる。各スケジューラは、方針 (policy) と機構 (mechanism) が分離されていて、方針を実行時に変更できるようになっている。

スケジューラを階層化することは、次のようなメリットがある。

- メタスペース毎に異なるスケジューリング方針を採用することができる。
- 一つのスケジューラが扱うアクティビティの数が減るので、スケジューリングの計算量が減る。

スケジューラの方針とは、各時刻においてどのオブジェクト (&レッド) を実行するかを決めるアルゴリズムのことである。これは通常、スケジューラの機構から呼び出される、あるいはスケジューラの機構を使う手続き/関数として実装される。リアルタイムという制限のない階層化スケジューラの場合は、各スケジューラは自分に割り当てられた CPU 時間をさらに自分の方針=アルゴリズムに従ってオブジェクトに割り当てればよく、問題はない。

4.2 リアルタイムスケジューリングの問題点

リアルタイムスケジューラを階層化した場合、2つの問題点がある。一つは、自分より上のメタ階層の (m^0に近い) スケジューラの種類によっては、デッドラインを満たせなくなってしまうことである。例えば極端な場合、1msec 間隔で周期的に実行しなければならないタスクがあったとき、メタメタ空間のスケジューラが 10msec のタイムスライスで動いていたら、メタ空間以下にどんなスケジュール方針を持ってきてもデッドラインを満たせない。これは、リアルタイムスケジューリングのアルゴリズムが、一つの CPU 全体のタスクを一度にスケジューリングすることを前提として考えられているからである。

また、別の問題点は、スケジュール可能性の判定に関するものである。リアルタイムでないスケジューリングの場合には、どのタスクを実行するかだけが重要で、その結果満たされるべき条件は特になかったので階層化も容易である。一方、リアルタイムの制限のもとでは、スケジュール可能性の判定も考慮しなければならず、これも階層化することが望まれる。つまり、スケジュール可能性の判定を、他のスケジューラの管理するタスクすべての情報ではなく、他のスケジューラが提供するより抽象化されたパラメタによって判断したい。

階層化スケジューラにリアルタイムの方針を持たせることは容易である。しかしそれだけではリアルタイムスケジューリングをすることはできない。リアルタイムスケジューリングで重要なことは、2章で述べたように、各タスクがデッドラインを満たすことである。つまり、スケジューリングアルゴリズムそのものよりは、それの満たす条件に注目している。そのため、条件を保ったまま Muse のオブジェクトモデルに組み入れる方法を考えなければならない。

4.3 Muse でのリアルタイムスケジューリング—方針の階層化

リアルタイムスケジューリングでは、方針を変えるといっても、満たすべき条件から来る制約上、大幅な変更はできない。どうしてもすべてのタスクのデッドラインを満たせない場合にどのタスクをあきらめるかという程度の変更しかない。つまりどのタスクから優先的にデッドラインを満たさせるかということが変えられればよい。

Muse では、リアルタイム性を要求するタス

通してサポートされる空間をメタ空間、m_m^2を通してサポートされる空間をオブジェクト空間と呼ぶ。

ク[5]は、最終的にメタメタ空間にある一つのリアルタイムスケジューラ (bf ベーススケジューラと呼ぶ) で管理される。周期の短いリアルタイムタスクの場合、固定優先順位なので簡単な RM を用いる。また、周期の長いリアルタイムタスクは、DL を用いる。そして余った時間をリアルタイムでないタスクのスケジューラに与える。しかしながら、あたかも階層化スケジューラがあるかのように見せるために、各リアルタイムスケジューラは、各タスクのパラメタから、自分の一つ上のメタ階層のスケジューラに与えるタスクのパラメタを生成して渡していく。要するに、ベーススケジューラ以外のリアルタイムスケジューラはフィルタのようなものである。

このパラメタは、次のものから成る。

重要度: デッドラインを満たせなかったときにどのタスクをあきらめるかを決めるのに使う。ベーススケジューラ以外のスケジューラにはそれぞれすでに上のメタ階層のスケジューラから重要度が割り当てられていて、管理するタスクの重要度はその範囲の中で決められる。

周期、計算時間: ベーススケジューラの方針が周期的タスクを想定しているために必要なパラメタである。

タスクの元々のパラメタは、必ずしもこの形をしている必要はないが、その意図を反映するように上のメタ階層のスケジューラに渡すパラメタを生成するのが、スケジューラの役割となる。そのため、このパラメタの生成アルゴリズムを意図した方針 **(intended policy)** と呼ぶ。

ここで採用したベーススケジューラの方針は、[Liu 73] で述べられている混合スケジューリングアルゴリズムであり、スケジュール可能性の評価が可能である。すべてのリアルタイムタスクは結局ベーススケジューラにスケジューリングされるので、全体のスケジュール可能性がわかることになる。ところが、この評価方法は複雑で、個々のタスクの情報が必要である。十分条件でよければ、混合スケジューリングは RM よりスケジュール可能性が高いので、RM の CPU 利用率の上限を用いて評価すれば、スケジュール可能性の評価も階層化することができる。

[5]Muse には、タスクという概念はないが、他のオブジェクトのメソッドを起動するのがサブルーチンコール的な意味ですむような、単純なアクティビティを便宜的にタスクと呼ぶことにする。

4.4 スケジューリングの例

次のようなタスクがあるとする。

A:周期的タスク (マウスドライバなど)			
タスク	周期	計算時間	重要度
1	10msec	4msec	3
2	1msec	0.05msec	5
B:非周期的タスク (RS-232C 非同期通信など)			
タスク	最短間隔	計算時間	重要度
3	2msec	0.1msec	4
4	4msec	0.5msec	4
C:タイムシェアリングタスク (通常のアプリケーション)			
タスク	計算時間		
5	500msec		
6	5msec		

A 、B 、C の各タスクグループは、それぞれ性格が異なるので、異なるスケジューラに管理させる。これらはメタメタ空間が直接管理しているとし、他のタスクのことは考えない。A 、B のスケジューラはリアルタイムスケジューラである。A のタスクは周期的なので、A の意図した方針により、そのままベーススケジューラにスケジュールさせる。B のタスクは周期的ではないが、最短間隔を周期とすれば最悪の場合のデッドラインやスケジュール可能性が見積もれるので、B の意図した方針により、最短間隔を周期としてスケジュールさせる。C のスケジューラは、リアルタイムではないので、自分に割り当てられた CPU 時間を各タスクに割り当てる。平均応答時間が短くなるように、計算時間の短いと思われるタスク (ここでは 6) を優先するものとしよう。ベーススケジューラは、周期が 3msec より短いタスク (2 と 3) を RM で、それで余った時間で 3msec 以上のタスク (1 と 4) を DL でスケジューリングするとする。

図 6 にスケジューリング結果を示す。意図した通りにスケジューリングされていることがわかる。

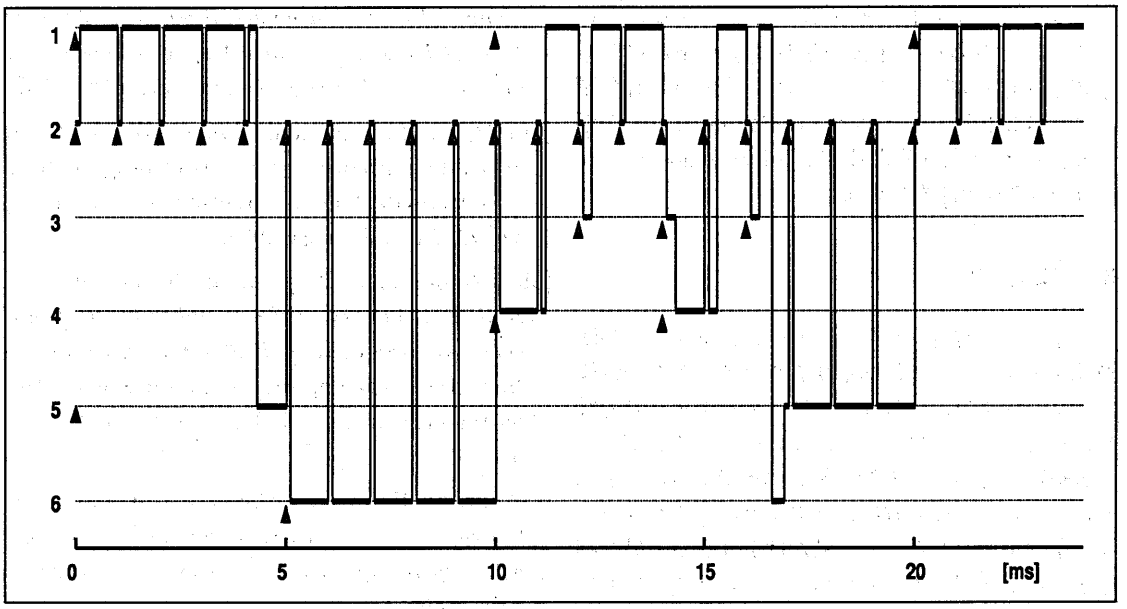

図 6: スケジューリングの例
▲はタスクの起動時刻であり、同時にデッドラインでもある。

5 実装方法

リアルタイム機能の実装方法を、注意するべき点を中心に述べる。

5.1 外部割り込み

リアルタイム性を保証するためには、リアルタイムタスクの実行中に不必要なコンテキストスイッチが起きないようにしなければならない。そのような要因となるのは外部割り込みである。そこで外部割り込みはすべて m^0 が受け、その割り込みで起動されるべきタスクがベーススケジューラによって CPU 時間を割り当てられた段階で、適当な m^n にメッセージとして送り、メソッドを起動することにする。実際には効率を上げるために、常に特定の m^n に制御を移すことが決まっている場合、m^0 は MuseCore に依頼して直接 m^n に制御を移すようにさせることもできる。

5.2 優先順位の継承

現在の Muse の実装では、オブジェクト毎に一つのスレッドがあるとしている。他のオブジェクトにメッセージを送ってメソッドを起動することは、他のスレッドの起動となる。多くの場合これは同期通信であり、サブルーチンコールの意味で行なわれる。すると、メッセージの送り先のオブジェクトの優先順位が低い場合、困った現象が起きる。例えば、優先順位が中間のオブジェクトが動いている場合、送り先のオブジェクトより優先され、送り元のオブジェクトが不本意に待たされることになる。これは、priority inversion[Sha 87] と呼ばれているものである。これを回避するためには、メッセージを送るたびに送り元の優先順位を送り先に継承しなければならない。

優先順位の継承は、オブジェクトの粒度が細かい場合、大きなオーバーヘッドになることが予想される。しかし、サブルーチンコールの意味でメソッドを起動することは、従来的な、オブジェクト間を移動するスレッドと同様である。そこで、粒度の細かいオブジェクトのサブルーチンコールの意味でのメソッドの起動は、実装上はサブルーチンコールで行い、優先順位を自動的に継承させることでオーバーヘッドを回避する。もちろん、リモートメッセージの場合には優先順位つきのメッセージ送出として実装されるが、この違いはユーザからは見えない。

オブジェクト毎に一つのスレッドしかないということは、さらに、複数の起動要求が来た場合も考慮しなければならない。これは従来の共有資源アクセスに相当し、やはり priority inversion が起きる。これを避けるために、priority inheritance protocol[Sha 87] を用いる。

6 まとめ

本稿では、Muse オペレーティングシステムの階層化スケジューラにおけるリアルタイム機能の位置付けは、個々のスケジューラにリアルタイムの方針を持たせることではなく、一つのベーススケジューラと複数の意図した方針であるべきであることを示した。このようにすれば、リアルタイムスケジューラの条件を満たしながら、見かけ上異なる方針を利用することができる。さらに、リアルタイム機能の実装方法を述べた。

現在のところ、タスク間に同期が必要な場合 (共有資源のある場合など) のスケジューリングアルゴリズムは考慮されていない。priority inversion を避けるための、basic priority inheritance protocol や priority ceiling protocol[Sha 87] を用いた場合にスケジューラが扱うべきパラメタを決定し、スケジュール可能性について考察する予定である。また、マルチプロセッサや分散環境でのスケジューリングを考慮する必要があるが、そのためにはリアルタイム通信を実現する必要がある。

リアルタイム機能の実装後は、ARTS のリアルタイムモニタ/デバッガである ARM[Tokuda 89] を使って、リアルタイムスケジューリングのモニタをし、評価の資料とする。

謝辞

本研究に対して数々の助言をいただいたカーネギーメロン大学計算機科学科の徳田英幸博士、および (株) ソニーコンピュータサイエンス研究所の諸研究員に感謝します。

References

[Lehoczky 87] John Lehoczky, Lui Sha, and Ye Ding. *The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior.* Technical Report, Department of Statistics, Carnegie Mellon University, 1987.

[Liu 73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, Vol.20, No.1, January 1973.

[Sha 87] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization.* Technical Report, Computer Science Department, Carnegie Mellon University, November 1987.

[Tokuda 89] Hideyuki Tokuda and Clifford W. Mercer. ARTS: A Distributed Real-Time Kernel. *Operating Systems Review*, Vol.23, No.3, July 1989.

[Yokote 89a] Yasuhiko Yokote, Fumio Teraoka, and Mario Tokoro. A Reflective Architecture for an Object-Oriented Distributed Operating System. In *Proceedings of European Conference on Object-Oriented Programming*, July 1989. also appeared in SCSL-TR-89-001 of Sony Computer Science Laboratory Inc.

[Yokote 89b] Yasuhiko Yokote, Fumio Teraoka, Masaki Yamada, Hiroshi Tezuka, and Mario Tokoro. The Design and Implementation of the Muse Object-Oriented Distributed Operating System. In *Proceedings of First Conference on Technology of Object-Oriented Languages and Systems*, November 1989. also appeared in SCSL-TR-89-010 of Sony Computer Science Laboratory Inc.

[Yokote 90] Yasuhiko Yokote, Fumio Teraoka, Atsushi Mitsuzawa, Nobuhisa Fujinami, and Mario Tokoro. The Muse Object Architecture: A New Operating System Structuring Concept. 1990. submitted to *Operating Systems Review.*

[Yonezawa 87]
Akinori Yonezawa and Mario Tokoro, editors. *Object-Oriented Concurrent Programming.* The MIT Press, 1987.