システムソフトウエアと
オペレーティング・システム 60−11
マルチメディア通信と分散処理 61−11
（1993. 7. 8）

Real-Time Mach3.0 における連続メディアサーバの実験 − QOS制御を取り入れたQuickTime Playerの評価 − †

舩渡 大地[1] 徳田 英幸[1,2]

[1] 慶應義塾大学環境情報学部
[2] カーネギーメロン大学計算機科学部

あらまし

本稿では、マルチメディア統合環境プロジェクトにおいて研究開発している分散マルチメディア環境を実現する基盤技術の中で、Real-Time Mach3.0 上での連続メディアサーバの実験に関して報告する。

まず連続メディアオブジェクトのモデルを紹介し、サービスの質 (QOS)、空間的解像度と時間的解像度、QOS クラス、QOS コントロールの概念を示す。さらに Real-Time Mach3.0 上でのリアルタイム処理の特徴について説明し、連続メディアサーバのプロトタイプとして、Real-Time Mach3.0 上で作成した QuickTime Player について述べる。最後に QuickTime Player の評価を通じて、リアルタイムスレッドを用いた QOS コントロールモデルの実証と、連続メディアサーバ構築に関する問題点を明らかにする。

キーワード　マルチメディアシステム、リアルタイムシステム 、分散リアルタイムカーネル、リアルタイムスレッド、マイクロカーネル、QOS、クイックタイム

Experiments with a Continuous Media Server in Real-Time Mach3.0

Daichi Funato[1] Hideyuki Tokuda[2,3]

[1]daichi@wide.sfc.keio.ac.jp Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan,
[2] hxt@sfc.keio.ac.jp, Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan,
[3] hxt@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract.

In this paper, we describe the evaluation of a prototype of continuous media server which has been implemented on top of the Real-Time Mach3.0 Microkernel.

We, first, introduce the notion of continuous media objects, quality of services (QOS), the spatial and the temporal resolution, QOS classes, and QOS control for continuous media objects. The real-time facilities of Real-Time Mach3.0 and the structure of the QuickTime player are described. We then show the evaluation results of the QuickTime player and summarize the issues of implementing continuous media servers which can guarantee a given QOS level.

Keywords: Multimedia System, Real-Time System, Distributed Real-Time Kernel, Real-Time Communication Protocol, Microkernel, QOS, QuickTime

†この研究は、情報処理振興事業協会 (IPA) が実施している開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトのもとに行なわれました。本稿に含まれている見解や結論は、著者自身のものであり IPA 自身の見解や結論を表すものではありません。

1 はじめに

従来まで、音声や動画など時間に従って変化していくデータを扱うマルチメディアシステムには、高価なハードウェアを使用した専用の独立環境が不可欠であった。しかし、高性能CPUや大容量メモリを備えたワークステーションやパーソナルコンピュータの普及に伴い、一般的なハードウェア環境下でのマルチメディアデータの取扱いが可能となってきている。

マルチメディア処理をコンピュータ上で行う本質的な意味は、以前は専用のハードウェアによって個別に処理されてきたデータを、デジタル技術によって統一的に取り扱うことにある。しかしながら、従来からのUNIXに代表されるタイムシェアリング方式に基づく資源管理を行った場合に、画像表示が一時的に止まってしまったり、音声が途切れてしまうといった不都合が起きている。

つまり、既存のコンピュータシステムはテキストを主とした時間的制約のないデータ処理を前提にしており、実時間性をもつビデオやオーディオなどマルチメディアデータに対しては、追加的な機能拡張によりアドホックに処理してきたといえる。

コンピュータ上でのマルチメディアシステムは動的に負荷が変化する。特に対話的なユーザ操作によってシステムが一時的に過負荷な状況(transient overload)に陥ってしまうことがある。そこで、サービスの質(QOS)を定義し、実行時にシステムの負荷や、アプリケーションの優先度に応じた資源割当てや実行単位のスケジュール制御を動的に行う必要がある。

マルチメディアデータを扱うためには、データ転送やバスアーキテクチャというハードウェアレベルから、プロトコルやOSといったソフトウェアレベルまで、あらゆるシステムの階層において時間とQOSの概念の導入が必要である。またさらに重要な点は、各階層間においてQOSの、負荷に応じた動的な制御にあると考えられる。

本稿では、まずこれらの時間的制約をもったマルチメディアデータとしての連続メディアオブジェクト(Continuous Media Object)の概念を紹介し、QOS(Quality of service)モデルの導入によるシステム構成を提案する。さらに、これらの連続メディアオブジェクトを用いた分散アプリケーションの一例として、Real-Time Mach3.0上でリアルタイムスレッドを用いたQuickTime Playerを実装した際に明らかになった問題点について指摘し、動的QOSコントロールモデルの実証や、今後の改良点について考察する。

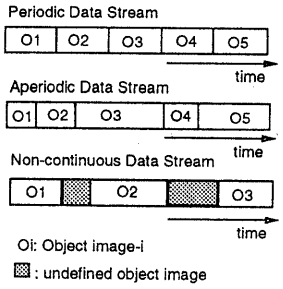

図 1: Continuous Media Objects

2 連続メディアのモデル

連続メディアオブジェクトは、通常のオブジェクトと異なり、デジタル化された音声やビデオデータなどのように時系列で連続的に変化するデータを呼び、オブジェクトの正当性が値の正当性だけでなく時間的な正当性にも依存している。

2.1 連続と非連続データストリーム

ここでは、連続メディアオブジェクトをデータストリームで表し、ストリームの構成要素である個々のデータオブジェクトに時間的制約が付随しているものと定義する。個々のデータオブジェクトに与えられた時間的制約が同一の場合、そのストリームを周期的データストリームと呼び、そうでない場合を非周期的データストリームと呼ぶ。また、データストリームが、連続的に定義されていない場合を非連続データストリームと呼ぶ。これらのデータストリームは図1のように表すことができる。

例えば、ビデオ、動画、音声データなどは、周期的データストリームで表わせ、オンラインのデスクトップ・プレゼンテーションは、非周期的データストリームで表すことができる。また、手動によるスライドなどのプレゼンテーションは、非連続データストリームとなる。

2.2 サービスの質(QOS)とクラス

連続メディアオブジェクトを扱うシステムにおいて、そのシステムが提供するサービスの質は、連続メディアオブジェクトに対する空間的解像度(spatial resolution)と時間的解像度(temporal resolution)によって表現することができる。例えば、ビデオや音声データなどの場合、時間的な解像度は、フレーム数/秒(fps)やサンプリングの速度で決定され、空間的な解像度は、データサイズやデータ圧縮率に依存する。

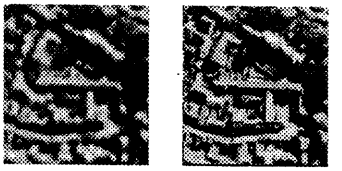

図 2: Spatial resolution of video image

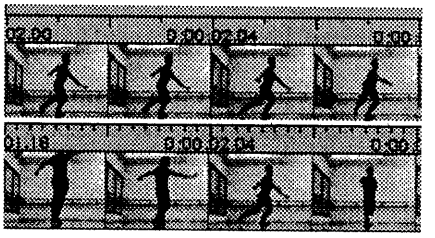

図 3: Temporal resolution of video image

空間的解像度の違いの例を図 2 に、時間的解像度の違いの例を図 3 にそれぞれ示す。図 2 でわかるように、高い空間的な解像度をもつデータオブジェクトの方がより精密にデータを表現できている。また、図 3 に示されているように、時間的な解像度が高いほど、より細かく連続的な動きを表すことが可能である。

また、実際のシステムにおいては、このような空間的、時間的解像度をユーザが細かく指定する代りに、いくつかの代表的な解像度を設定しておき、それらのクラス (QOS クラスと呼ぶ) を選択することを行っている。例えば、ビデオデータなどの場合、ユーザは、単に 1, 2, 3, 5, 10, 15, 30 fps (frame per second) といった QOS クラスから時間的解像度を選択したり、8, 16, 24 bpp (bits per pixel) といった QOS クラスから、空間的解像度を選択している。

ユーザから与えられたこれらの QOS クラスをもとに、システム資源の利用状況に応じて、動的にビデオセッションなどの QOS をコントロールする方式も開発している [9]。

3 Real-Time Mach3.0

Real-Time Mach 3.0 は、カーネギーメロン大学で開発された Mach 3.0 マイクロカーネル [1, 4] をベースに ARTS カーネルで開発したリアルタイムスレッド、リアルタイムスケジューラ、同期操作、タイマオブジェクト、クロックオブジェクト、リアルタイムプロトコルモジュールなどを取り入れてリアルタイム処理用に拡張したマイクロカーネルである。従来のリアルタイムエグゼクティブと違い、解析でき、かつ予測可能な分散リアルタイム OS を提供することを目標としている [7, 8]。また、昨年度から慶應義塾大学環境情報学部でのマルチメディア統合環境プロジェクトにおいても、分散マルチメディア環境を支える共通基盤ソフトウェアのためのカーネルとして、Real-Time Mach にマルチメディア対応のための機能拡張が行なわれてきている [6, 10]。

Real-Time Mach3.0 上では、Mach 3.0 と同じ UNIX サーバ [4] や X ウインドウシステムを動作させることができる。つまり、リアルタイム用のプログラムとともに、従来の Mach 用に作成されたプログラムも同時に混在して動作させることができる環境を提供している。また、Real-Time Server(RTS) や Network Protocol Server (NPS)[5] を用いることで UNIX サーバのオーバーヘッドなしに、分散リアルタイム・アプリケーションを直接マイクロカーネル上で動作させることもできるようになっている。以下では、リアルタイムスケジューラとリアルタイムスレッド [8] の概要を述べる。

3.1 リアルタイムスケジューラ

一般に、リアルタイム OS で採用されているスケジューリングの技法は、そのシステムが対象としているリアルタイムタスクの性質、すなわち、静的／動的なタスクセット、周期／非周期的なタスクなどの状況によって大きく左右される。従来からの組み込み型システムでは、サイクリック・エグゼクティブモデル (Cyclic Executive Model) が多く使われているが、タスクセットが複雑になったり、タスクの変更が頻繁に起こる場合は、割り当て作業を手動で行ない、再構成をしなければならないなど多くの問題を含んでいた。

Real-Time Mach では、レート・モノトニック方式や最短デッドライン優先方式などリアルタイムタスクを考慮しつつ、従来の TSS 型のスケジューリングポリシーもサポートできるといったスケジューリングポリシー選択可能なスケジューラが実現されている。例えば、レート・モノトニック方式は、周期の一番短いタスクに最高の優先順序を与える横取り可能なスケジューリング方式であり、従来のモデルと違いスケジューラビリティの解析システムなどとの整合性が非常に良いのが特徴である [8]。特に、周期的データストリームを扱う周期的なアクティビティの処理において有効な方式である。現在 Real-Time Mach でサポートされているスケジューリング・ポリシィ

は次のとおりである。

Mach Timesharing: Machの時分割スケジューリング

Fixed Priority/RR: 固定プライオリティ/ラウンドロビン方式スケジューリング

Fixed Priority/FIFO: 固定プライオリティ/FIFO方式スケジューリング

Rate Monotonic: 周期の一番短いスレッドに最高の優先順序を与える先取り可能なスケジューリング

Deadline Monotonic: デッドラインの短いスレッドに最高の優先順序を与える先取り可能なスケジューリング

Earliest Deadline First: デッドラインに最も近いスレッドから順に実行権を与える先取り可能なスケジューリング

また、これらのポリシーは、各プロセッサセット (i.e., プロセッサとスレッドの集合) ごとの属性として設定されており、processor_set_get_attribute( ) と processor_set_set_attribute( ) のプリミティブを使って変更することが可能である。

3.2 リアルタイムスレッド

従来から用いられている計算のモデルは、プロセス・モデルである。多くの商用リアルタイム・エグゼクティブでは、単純なマルチ・プロセスのモデルを採用しており、プログラムの動作に対する時間的制約は、プログラム上には明示されていなかった。従って、実行時のプロセッサスピードが数倍に速くなった場合など、アニメーションといった連続データの表示などが、プロセッサスピードに比例して速くなってしまうという問題も起きていた。

一方、より "予測可能な" リアルタイムシステムを目指す、新しい分散リアルタイムシステムでは、プログラムの動作に対する時間的制約をプログラム上で明示する方法[1]が取り入れられてきている。

また、マルチ・プロセスモデルの欠点のひとつであったコンテキスト・スイッチングのオーバヘッドを減少させるためや、並列プロセッサとの整合性をよくするために、並行プロセスの動作を "スレッド" で記述し、スケジューリングの単位として採用してきている計算のモデルが開発されてきている 。

Real-Time Mach では、ARTS カーネルで開発したリアルタイムスレッドの概念と Mach の "C-Thread" パッケージ [2] を融合させた RT-Thread モデルが提供されている。Real-Time Mach では、次のようなリアルタイムスレッドを制御するシステムプリミティブがマイクロカーネルに実装されている。

rt_thread_create(task, thread, thread_attr)
スレッドを生成する。
rt_thread_exit(thread)
スレッドを終了する。
thread_get_attribute(thread, flavor, old_attr, old_attrCnt)
スレッドの属性を読み出す。
thread_set_attribute(thread, flavor, new_attr, new_attrCnt)
スレッドの属性をセットする。

RT-Thread のインタフェースは、基本的には、C-Thread の拡張であるが、リアルタイムタスクにおいて、より正確に周期的スレッドを記述ができ、かつデッドライン等の時間的的制約を明示できることが特徴である。Real-Time Mach では、時間的制約をリアルタイムスレッドの時間属性として、動的に与えることができる。例えば、周期的なスレッドの場合、開始時刻、周期、周期内のオフセット、デッドライン、アボートタイム、最悪実行時間、スレッドの重要度などを指定する。また、非周期的なスレッドの場合、デッドライン、アボートタイム、最悪実行時間、スレッドの重要度などとともに、最悪到着待ち時間を指定する。これらのリアルタイムスレッドの時間属性データを参考に、ハード／ソフトリアルタスク両方のサポートが可能である。

4 QuickTime の概要

まず、今回の QOS 実験のサンプルとして採用した QuickTime について説明する。QuickTime[13] は、Apple 社が Macintosh 用に開発した、マルチメディアに代表される時間ベースの情報を扱うための拡張システムソフトウェアである。

従来の Macintosh 上において、動画と音の同期や圧縮が必要なアプリケーションは、すべて独自の方法を用いて管理していたが、システムソフトである QuickTime の出現により、それらの管理に標準的なアクセスを提供することが可能となった。

QuickTime では、動画と音の同期を時間軸を基準にして管理しており、Movie Toolbox や Image Compression Manager を始めとしたモジュールの集合体として提供されている。さらに、各種のハードウェアの制御や様々な圧縮方式が取り扱える Component Manager のモジュール化により、様々なタイプのマルチメディア情報に対する抽象化が図られ、柔軟な機能拡張が可能となっている。

QuickTime がサポートする標準的なデータはムービー形式である。ムービー形式は 1 つ以上のトラックと呼ばれるデータストリームを含んでおり、トラックは実際のデータが保存されているメディアをアクセスする。

複数のトラックを含むムービー形式のイメージを図 4 にしめすが、各トラックに含まれるのは実際のデータストリームではなく、メディア参照のためのリストである。

[1] 従来の方法を、プログラムと時間的制約の implicit binding と言い、明示的な方法は、explicit binding という。

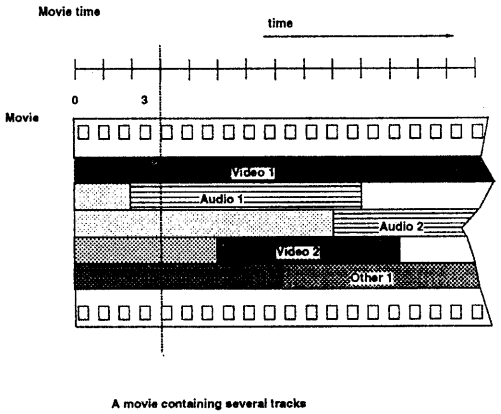

図 4: 複数のトラックを含むムービーのイメージ

4.1 QuickTime における時間管理

QuickTime では連続メディアデータの時間的な管理機構として、各メディアがタイムコーディネイトシステムを保持している。タイムコーディネイトシステムでは、タイム スケールと持続期間を定義しており、タイムスケールは 1 秒あたりのサンプル数、持続期間はムービーあるいはメディアのサンプルの合計を表している。

特定の時間のサンプルを取り出すには、スタート時を 0 とした経過時間を、タイムスケールを基準にした現在のサンプル番号に変換すればよい。つまり、$CurrentSample = Timescale * (Systemclock - Starttime)$ で算出する事が出来る。通常、オーディオ情報はスキップせずに出力されなければならないため、この機構は主にビデオフレームのスキップに用いられる。

各メディア間のタイム スケールの差は Movie Toolbox がムービーのタイムコーディネイトシステムから各メディアのタイムコーディネイトシステムにマップすることで解決している。

これらの機構により、システムが過負荷になり処理が遅れた際でも、現時点のサンプルを取り出すことで時間的な不整合に対しての処理が可能である。

しかしながら、この様な時間管理方式は、時間のタイミングに関する整合性を主眼としており、フレームレートの確保など、連続メディアデータのサービスの品質保証に関しては考慮されていない。

5 QuickTime Player の実装

マルチタスク、および分散マルチメディア環境下で連続メディアをサポートするには、実時間性の保証が必要である。さらに、システムの負荷の変動によりシステムの処理がアプリケーションの要求に追い付かない場合は、空間的解像度、あるいは時間的解像度の制御によってサービスの質 (QOS) を動的に変化させる必要がある.

今回実装した QuickTime Player では、Real-Time Mach 3.0 に実現されているリアルタイムスレッドを用いることにより基本レベルの実時間性を確保し、アプリケーションレベルとカーネルの協調作業による動的な QOS 制御の実験を行った。

実際に作成した QuickTime Player は、時間的な QOS 比較実験のために 3 種類ある。まず、リアルタイムスレッドを使わずに、Best effort で再生を続けるもの。次に、リアルタイムスレッドを使い、指定された QOS を守りながら再生をするもの。最後に、セルフスタビライズ機能をもたせ、マルチセッションによるシステムの負荷に応じて、自ら QOS のレベルをさげるもの、の 3 つである。

5.1 UNIX 上での QuickTime Player

Macintosh 上で作成された QuickTime のデータを UNIX 上で解釈するには、ムービーのデータフォーマットを格納しているムービーリソースを参照しなくてはならない。

ムービーリソースは Macintosh ファイルシステム中のリソース部に保存されているため、ファイルを UNIX 上に転送した上でリソース部とデータ部に分割し、リソース部に記述されているデータ構造を読み込んで、データ部に収められているイメージデータを取り出し、描画する必要がある。

このようにムービーは、図 5 に示すように複雑な入れ子になった構造体の集合として定義されており、データ部に格納されているイメージのオフセット値やサイズ、タイムスケールや持続時間などの情報が含まれている。この情報を読みこむことで、UNIX 上でも QuickTime の再生が可能である。

5.2 Best Effort 方式の Player

今回まず最初に作成したのは、QOS のレベル、システムの負荷等の状況は一切考慮せず、スケジューリングされた時点で全力で回り続ける、いわゆる Best effort のプログラムである。

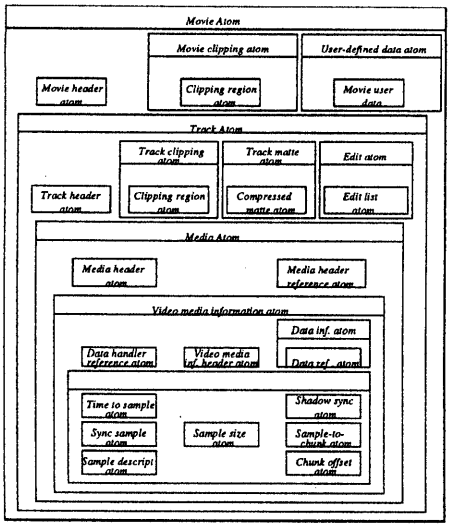

図 5: ムービー構造体の例

5.3 Real-Time Mach3.0 への拡張

Best effort のプログラムでは、システムの負荷の増減によって時間的な解像度が変化してしまう。そのため、Best effort のプログラムを周期的なスレッドとデッドラインハンドラを用いた Real-Time Mach3.0 版の QuickTime Player へ拡張した。

QOS 保証の実験のため、プログラム起動時にフレームレイト (fps) を指定し、その値を周期スレッドの実行周期属性値に変換した上で、RT-Thread のライブラリコールである rt_thread_attribute_init() を使って RT-Thread 属性に設定をした。デッドラインハンドラを使った制御は、本方式では行わない。

5.4 セルフスタビライズ機構の利用

リアルタイムスレッドの周期を指定しただけのプログラムでは、動的に変動するシステムの負荷に対応することが出来ない。

そのため、第 3 の拡張としてシステムの負荷に応じて動的に QOS を変更するセルフスタビライズ機構を備えた QuickTime Player を実現した。

rt_thread_create() を使い、デッドラインハンドラ・スレッドと周期スレッドを起動する。この際、プログラムの本体は、周期スレッドの終了まで sleep している。周期スレッドがデッドラインを満たせなかった場合、周期スレッドのデッドラインポートを通じデッドライハンドラへメッセージが送られる。デッドラインポートで expire_wait() を呼び、メッセージが送られてくるのを待っていたデッドラインハンドラは、適当な処理をした上で thread_resume() を行い、周期スレッドの実行を再開させる。

周期スレッドがデッドラインをミスすると、起動されたデッドライハンドラはデッドラインミスのフラグを立てる。そして、周期スレッド自身の中で RT-Thread 属性変更のためのルーチンを置き、規定回数デッドラインミスが起こっているなら、周期スレッドが自分の周期属性値を増やした上で thread_set_attribute() を呼び、RT-Thread 属性の再設定を行う。デッドラインミスが生じていない場合は、システムに余裕があると見なし、同じように RT-Thread 属性変更のルーチンを呼び、周期属性値を減らした後、thread_set_attribute() により、自分の属性の再設定をおこなう。

このようにして、システムの負荷に応じて自分自身の周期をコントロールする自律的なスレッドを作り出すことができる。

6 QuickTime Player による QOS の実験とその評価

今回の実験では、CPU スケジューリングと連続メディアデータの問題に主眼を置いたので、ディスクアクセスを行わず、起動時にイメージを読み込み、メモリにマップした状態で測定を行った。イメージの大きさは 160*120*8/8=19200byte であり、読み込んだイメージ数は 20 フレーム分である。

実験は、システムに余計なタスクがない状態で、2 通りのスケジューリングポリシーによる Best effort 方式と Real-Time thread の時間的 QOS の安定度の比較を行い、同時に、異なるスケジューリングポリシーでの連続メディアデータの時間的 QOS の安定度を測定した。

次に静的に時間的 QOS のレベルを指定したリアルタイムスレッドを起動し、その限界について測定をした。

最後に、複数のセッションを起動することにより、システムの負荷に応じて動的に時間的 QOS のコントロールを行うリアルタイムスレッドの実験を行った。

評価には Intel i486DX2/66MHz CPU を搭載した IBM PC AT 互換機 Gateway 2000 4DX2-66V で 250 ナノ秒精度のタイマボードを使って測定した。Real-Time Mach3.0 はバージョン MK78、UNIX サーバはバージョン UX39 を使用した。

6.1 スケジューリングポリシーによる QOS 安定度の比較

マルチメディアアプリケーションでは、スレッドの周期性が大切である。そのため、スケジューリングポリシーの違いによる周期の乱れを測定した。ここでは、Mach Timesharing 方式と Rate Monotonic 方式について行った。システムに余計な負荷がかからないように、余分な

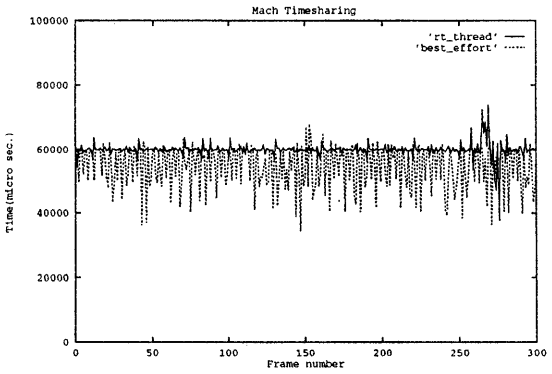

図 6: Mach Timesharing

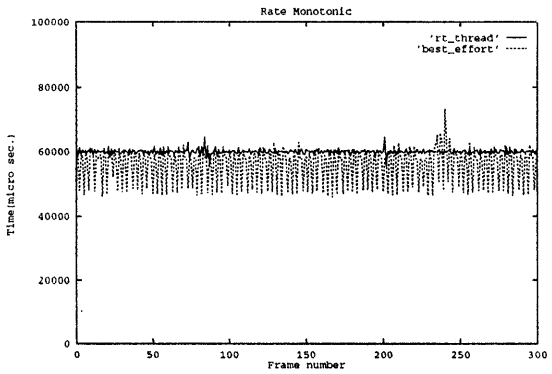

図 7: Rate Monotonic

タスクを排除した上で、Best effort と Real-Time thread の双方について測定してみた。周期が正確にわかるように、1 フレーム毎にかかった時間を測定した。

結果を図 6,7 に示す。外乱が少ない状況においても、Best effort、Real-Time thread 双方において Rate Monotonic のほうが安定した周期で起動されることがわかる。

6.2 時間的解像度の保証

次に、Real-Time thread を使った QuickTime Player を用いて、時間的な QOS の保証の実験を行った。QOS の時間的解像度としては、1 秒あたり 5/10/15/20 フレームのクラスを試した。

実験結果は図 8 に示すように、5/10/15 フレームのクラスでは安定したフレーム数が得られており、デッドラインミスの回数も全て 0 であった。しかし、システムの性能の上限を超えていると思われる 20 フレームのクラスでは、起動直後は何とか指定したフレーム数を得られていたが、いったんデッドラインをミスすると、ハンドラに飛ぶ分だけコストが余計にかかり、もう元のフレーム数が得られなくなっていることがわかる。20 フレームクラスの場合のデッドラインミスは、全体の 65%に達した。このような静的な指定のみでは QOS の保証が出来ず、複数のアプリケーション間での協調作業は不可能である。

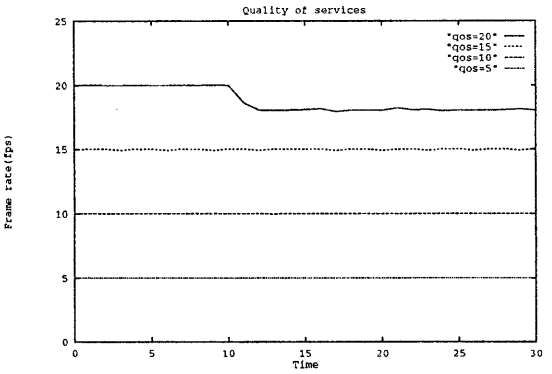

図 8: 時間的 QOS の保証

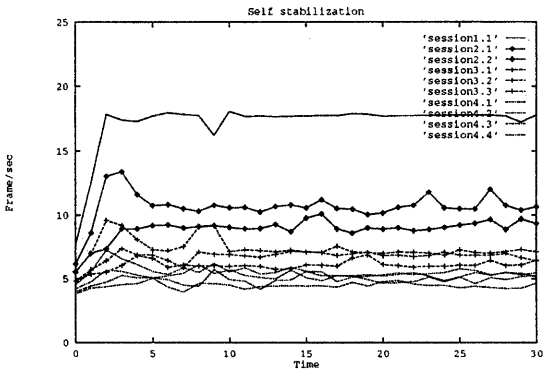

図 9: セルフスタビライズ機構

6.3 セルフスタビライズ機構によるマルチセッションの制御

複数のセッション間で、相互に動的に時間的解像度を変更した実験結果を図 9 に示す。

今回の実装では、連続 2 回デッドラインハンドラが呼ばれた時点で 10 ミリ秒周期を遅らせ、連続 4 回周期を満たした時点で 5 ミリ秒周期を縮めるようなアルゴリズムをとった。

1 セッションの場合は、18fps にほぼ安定して収束している事がわかる。2 セッションの場合は、9〜10fps の間へ収束している。3 セッションの場合は、6〜7fps の間へ収束している。4 セッションの場合は、4〜6fps の間へ収束している。それぞれのセッション毎の収束にばらつき

があるが、どの場合でも、各セッションの1スレッド毎に観察すると、安定したフレーム数を出している事がわかる。

この様なアプリケーションとカーネルとの協調作業を利用すれば、メディアサーバがある一定の要求されたQOSを満たせない場合に、他のクライアントのQOSを許容範囲で下げたり、また、クライアントの新規参入を排除するAdmission Controlのような制御と統合することにより、より高度な動的QOS制御が可能となる。

7 まとめ

本稿では分散マルチメディア環境を実現する基盤技術の中で、連続メディアサーバについて実験を行った。

今回の実験では、イメージを予めメモリに読み込んでから描画を行った。ハードディスクのアクセス時間のばらつきが、実験結果にそのまま反映されてしまうからである。動画や音声の再生のためには、データ読み出しの遅延に対する時間制約を持たせなくてはならない。同じ事は今回の場合のXウインドウシステムやUXサーバについてもいえる。カーネル側でリアルタイム性を保証しても、描画やシステムサーバの側でプライオリティインバージョンが起きてしまえば、時間的制約は満たされない。つまり、リアルタイム性を持ったマルチメディア環境を実現するには、あらゆる階層での実時間性を実現しなくてはならない事がわかる。

OSやネットワーク、バスにいたるレベルまで、マルチメディアデータを扱うための様々な研究がなされている。その中で、今後の高度なリアルタイム性をもった分散システムを開発するためには、それぞれの階層をつなぎ合わせ、実時間性とサービスの質を保証するQOSの技術が重要になると考えられる。

8 謝辞

本プロジェクトを遂行しするにあたり、協力して頂いたカーネギーメロン大学のARTグループ、Machグループの皆様、慶應義塾大学の開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトの皆様に感謝致します。さらに、御指導戴いた慶應義塾大学環境情報学部の斎藤信男教授に感謝致します。

参考文献

[1] M.J. Accetta, W. Baron, R.V. Bolosky, D.B. Golub, R.F. Rashid, A. Tevanian, and M.W. Young, "Mach: A new kernel foundation for unix development", *In Proceedings of the Summer Usenix Conference,*, July, 1986.

[2] E. C. Cooper, and R. P. Draves, "C threads", *Technical report, Computer Science Department, Carnegie Mellon University, CMU-CS-88-154*, March, 1987.

[3] T. Fisher: "Real-Time Scheduling Support in Ultrix-4.2 for Multimedia Communication," *In Proc. of the 3rd International Workshop on Network and Operation System Support for Digital Audio and Video, pp.282-288*, 1992

[4] D. Golub, R. Dean, A. Forin, and R. Rashid, "Unix as an application program", *In the proceedings of Summer Usenix Conference*, June, 1990.

[5] T. Nakajima, T. Kitayama and H. Tokuda, "Experiments with Real-Time Servers in Real-Time Mach," In *Proc. of 3rd USENIX Mach Symposium*, Arp. 1993.

[6] 緒方 正暢, 人見 潔, 和田 英彦, 追川 修一, 徳田 英幸, "Real-Time Mach 3.0 の評価と改良" In *Proc. of RTP '93*, 情報処理学会/電子情報通信学会, Mar. 1993.

[7] S. Savage and H. Tokuda: "RT-Mach Timers: Exporting Time to the User," In *Proc. of 3rd USENIX Mach Symposium*, Apr. 1993.

[8] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", In *Proceedings of USENIX Mach Workshop*, October, 1990.

[9] H. Tokuda, Y. Tobe, S.T.-C. Chou and J. M. F. Moura, "Continuous Media Communication with Dynamic QOS Control Using ARTS with and FDDI Network," In *Proceedings of ACM SIGCOMM '92*, August, 1992.

[10] 徳田 英幸, 斎藤信男, "マルチメディア統合環境プロジェクトにおけるリアルタイム処理技術," *In Proc. of RTP '93*, 情報処理学会/電子情報通信学会, Mar. 1993.

[11] K. Nichols, "Performance Studies of Digital Video in a Client/Server Environment," *Third International workshops on Network and Operation system support for digital audio and video*, Nov, 1992.

[12] L. Lowe "A Continuouos Media Player," *Third International workshops on Network and Operation system 11 support for digital audio and video*, Nov, 1992.

[13] "QuickTime 1.5 Developer Kit," *Inside Macintosh: QuickTime*,Apple Computer, Inc. 1992.