オペレーティング・システム 53－4
（1991. 12. 6）

# Mach上でのリアルタイムイベント通知の実現

中島 淳、矢崎昌朋、松本 均

(株) 富士通研究所
ヒューマンインタフェース研究部

マルチメディアアプリケーションをサポートするために、Mach オペレーティングシステムを拡張した。マルチメディアのうち、特にアニメーション、音声、音楽などの連続メディア (continuous media) をサポートする。この拡張は、1. リアルタイムイベント通知とそのスケジューリング、2. Temporal paging system、3. ユーザモードのデバイスドライバから構成される。各ハンドラをイベントの通知によって起動し、時間的な制約を満足するようにスケジューリングすることによって、マルチメディアアプリケーションをサポートする。temporal paging system は、巨大なデータ領域を持つマルチメディアアプリケーションのリアルタイム性をサポートすることを目的としている。さらに、ユーザモードのデバイスドライバはカーネルを再構築せずにデバイスドライバを追加する機能を提供する。拡張の概要、実現、そして評価について触れる。

# Realtime Asynchronous Event Notification on the Mach Operating System

Jun Nakajima, Masatomo Yazaki, Hitoshi Matsumoto

Human Interface Laboratry
FUJITSU LABORATORIES, Ltd.

We have extended the Mach operating system to support multimedia applications. This paper focuses on realtime issues while supporting continuous media, such as animated graphics, sound and music: asynchronous event notification, temporal paging system, and device drivers that run in user mode. Our extensions support multimedia applications by notifying occurrences of events to the handlers and schedule them to meet their time constraints. Temporal paging system is designed to support realtime handling for multimedia applications that have huge data segments. User-mode device drivers will help develop device drivers efficiently and provide a flexible environment. The rationale, design, implementation and performance are presented.

## 1 はじめに

マルチメディアはヒューマンインタフェースの向上だけでなく、計算機利用の新しい可能性を提供する。マルチメディアアプリケーションを開発環境のすぐれた汎用OSでサポートすることは、マルチメディアアプリケーションを発展させる上で不可欠なことである。

今回、われわれはマルチメディアアプリケーションをサポートするために、Mach オペレーティングシステム [1] を拡張した。この拡張 (Multimedia/Realtime Extensions) はマルチメディアのうち、特に、アニメーション、音声、音楽などの連続メディア (continuous media) をサポートする [2]。一般のマルチメディアアプリケーションをサポートするためには、この他に様々な機能が必要になるが、連続メディアのサポートはOS自体に必須な機能として考えたからである。また、カーネルを再構築せずにデバイスドライバを追加する機能も含めた。

連続メディアをサポートするためには、OSにリアルタイム性が必要である。通常、連続メディアは、周期的な処理を繰り返すことによって実現される。したがって、時間的な制約を満足しないと、メディアの内容が異なったり、品質が劣化してしまう。また、マルチメディアのアプリケーションはいくつかのメディアを同期させることがあるので、リアルタイムなメディアの同期をサポートする必要がある。

複数のメディアを組み合わせるマルチメディアアプリケーションは、単一の制御構造で実現するよりも、それぞれの異なるメディア別に制御構造を用意し、それらの間で同期をさせた方が実現が容易である。複数の制御構造でそのようなアプリケーションを実現した場合、特に重要なのは同期の正確さである。しかし、一般にUNIX などのシグナルによるプロセス間の同期は、リアルタイム性を保証しない。このためにPOSIX では、realtime signal が提案されている [3]。

Mach はマルチメディアアプリケーションをサポートする素質を持っている。スレッドという制御単位をサポートしており、一つのプロセス (正確にはタスク) 内に複数の制御構造を用意できる。ただし、現状のスケジューリングは、通常の UNIX と同じタイムシェアリングのポリシーを使うので、リアルタイムなメディアの同期をサポートできない。

また、Mach3.0、マイクロカーネルは、リアルタイムをサポートするのに有利である。UNIX はサーバとしてユーザモードで動くので、プリエンプティブである [4]。UNIX のシステムコールを実行中でもリアルタイム性を要求する他のスレッドを優先することができる。したがって、カーネルモードでの遅延が減少する。以上がマルチメディアアプリケーションをサポートするために Mach をプラットフォームとして選択した理由である。

## 2 Multimedia/Realtime 拡張の概要

われわれの拡張は、次のものから構成される。

- リアルタイムイベント通知と preemptive deadline-driven スケジューリング
- Temporal paging system
- ユーザモードのデバイスドライバ

リアルタイムイベント通知は、リアルタイムなスレッドの起動と同期を可能にし、連続メディアとリアルタイムなメディアの同期をサポートする。また、preemptive deadline-driven スケジューリングという、デッドラインスケジューリングをマルチメディアデバイス用に改良したスケジューリングを用いて、リアルタイムなスレッドのスケジューリングを行なう。

巨大なデータ領域を持つマルチメディアアプリケーションのリアルタイム性をサポートすることは難しい。仮想記憶をサポートする OS の場合、頻繁にアクセスされない領域はスワップアウトの対象になる可能性が高い。イベント通知が起きた時に、二次記憶から物理ページにスワップインされる時間の上限は予測できない

[5]。テキスト領域は、一般的に静的に決まるため、予め物理ページに pin-down するのは比較的容易である。データセグメントは動的に変化する。巨大なデータ領域を pin-down するは現実的でない場合がある。そのために、temporal paging system という方法によって、アドレスと時間の二次元空間で paging を行なうことにより、これを解決する。

また、マルチメディアに対応させるためには、OS は様々なデバイスドライバを用意しなければならない。マルチメディア用のデバイスはこれまでの伝統的なデバイスとはかなり異なっている。また、様々なメディア用のデバイスが次々と開発されている。したがって、カーネルを再構築せずにデバイスドライバを追加する機能が必要である。

なお、ユーザモードのデバイスドライバの詳細ついては、[2] を参照願いたい。

## 2.1 プログラミングモデル

マルチメディアアプリケーションは、それぞれのメディア別に制御構造を用意し、それらを協調動作させることによって実現できる。それぞれの制御構造をハンドラと呼ぶ。われわれの仮定するプログラミングモデルは、以下のようである。

- ハンドラはイベントの通知によって起動される。
- ハンドラは並行に動作できる。
- ハンドラは時間的な制約を持つ。

連続メディアを実現する場合、ハンドラは時間的な制約を満足しなければならない。ただし、これはそれぞれのメディアの性質で決まるのではなく、それを扱うデバイスによって決まることに注意しなければならない（2.3 参照）。

## 2.2 イベントとイベント通知

この拡張でのイベントは、以下のものである：1. 非同期 I/O の終了、2. タイマの発火、3. ユーザ定義のイベント、4. マルチメディアデバイスからの割り込み。それぞれのイベントは、ユニークなクラスを持つ。一つのイベントが起こると、関係のあるすべてのハンドラにイベントが通知（以下、イベント通知と呼ぶ）され、それらのハンドラは起動される。例えば、MIDI (musical instrument digital interface) によって音楽を演奏する場合、タイマをある時間周期で発火させて MIDI ハンドラを起動し、MIDI コマンドを送ることによって実現できる。これとグラフィクスを同期させる場合には、グラフィクスのハンドラを用意し、同様にタイマイベントを通知させればよい。

ユーザが与えるイベントの定義（event definition structure）を、図 1に示す。*evt_handler* はイベン

```
void            (*evt_handler)();
void            *evt_value;
event_class_t   evt_class;
event_set_t     evt_classmask;
event_opt_t     *evt_option;
```

図 1: イベント定義の構造体

ト通知によって起動されるハンドラのエントリポイント、*evt_value* はイベント通知が起きた時にアプリケー

ションに渡される値、*evt_class* はハンドラがどのクラスによって、起動されるかを指定し、*evt_classmask* は、そのハンドラが実行している間ブロックするクラスを示すマスクである。これらの定義は、POSIX の前提案 [3] に準拠するものである。

ハードウェアデバイスからの割り込みをアプリケーションにリアルタイムで通知することにより、柔軟性、対話性のあるアプリケーションを作成することができる。

## 2.3 マルチメディアデバイスのタイプと時間的な制約

連続メディアの時間的な制約はデバイスによって決まる。ハードウェアデバイスの進歩によって時間的な制約をハンドラが考慮する必要がなくなるかも知れないが、現状ではいくつかの時間的な制約がある。

マルチメディアデバイスの時間的な制約は、デッドラインとレスポンスタイムによって決まると仮定する。この仮定に基づき、マルチメディアデバイスのタイプを次の二つの種類に分けた。

- デッドライン駆動 (deadline-driven) — バッファをフラッシュすることによって連続メディアを実現する場合、フラッシュするデッドラインに間に合えば、そのメディアの品質は劣化しない。つまり、デッドラインより以前にバッファにデータ用意しても結果は同じである。
- イベント駆動 (event-driven) — ある絶対的な時間にデータを入出力することにより連続メディアを実現する場合、レスポンスタイムが遅れるにつれ、メディアの品質は劣化する可能性がある。

デッドライン駆動型、イベント駆動型を使うハンドラをそれぞれ、デッドライン駆動型、イベント駆動型ハンドラと呼ぶ。

## 2.4 Preemptive Deadline-Driven スケジューリング

商用のリアルタイム OS では、プライオリティをそれぞれのハンドラに割り当てるものが多いが、時間的な制約をプライオリティにマッピングすることは難しい [6]。優先順位を動的に割り当てる場合、deadline-driven スケジューリングはシングルプロセッサシステムでは高い CPU 使用率を獲得できる [7]。

複数のデッドライン駆動型、イベント駆動型ハンドラが並行に動作する場合の、適切なスケジューリング・ポリシーの一つは以下のようである。

1. イベント駆動型ハンドラをなるべく先送りしつつ、
2. デッドライン駆動型ハンドラのデッドラインを守る。

このポリシーを持つスケジューリングを preemptive deadline-driven スケジューリングと呼ぶ。イベント駆動型ハンドラを先送りすることは、deadline-driven スケジューリングの能力を低下させることになる。しかし、以下の仮定をすれば、深刻な低下にはならない。すなわち、一般的なイベント駆動型ハンドラはバッファを持たず（あるいは、かなり小さい）、短い時間内に処理が終了するということを仮定する。

Preemptive deadline-driven スケジューリングは、時間的な制約以外にイベントの相対的な重要度を必要とする。限られた CPU 資源でリアルタイムスケジューリングを行なう場合、優先順位の割り当てが常に可能であるとは限らない。preemptive deadline-driven スケジューリングの場合、次のようなことが起こる。

- イベント駆動型ハンドラを先送りすると、デッドライン駆動型ハンドラのデッドラインを守ることができなくなる。

例えば、1. スケジューラが 10ms のタイムスライスで動き、2. デッドライン駆動型のデッドラインまでの時間が 10ms より小さく、3. イベント駆動型ハンドラの処理時間が 10ms よりも大きい場合、イベント駆動型ハンドラを先送りしたら、デッドライン駆動型のハンドラのデッドラインを守ることが明かにできなくなる。この場合の選択として二つある。すなわち、

1. イベント駆動型ハンドラを先送りせずに、デッドライン駆動型のハンドラを先に実行する。

2. イベント駆動型ハンドラを先送りする。デッドライン駆動型のハンドラは、デッドラインに間に合わなくなるかもしれない。

これらのどちらか選ぶべきは、アプリケーション、メディアによってきまる。ヒューマンインタフェースの点から考察した場合、動画などのように1回のバッファのフラッシュが行なわれなくても人間にはわからない場合もある。したがって、これらの選択の指針は、アプリケーションによって与えられるべきである。

Preemptive deadline-driven スケジューリングのために必要な定義、*evt_option* (event option definition structure) を図2に示す。

```
timespec_t      evt_dtime;
timespec_t      evt_etime;
int             evt_weight;
event_type_t    evt_type;
```

図 2: Preemptive deadline-driven スケジューリングのための構造体

*evt_dtime* はイベントの発生からデッドラインまでの時間、*evt_etime* は最悪の場合のハンドラの実行時間、*evt_weight* は、そのハンドラの重要度、*evt_type* はイベントのタイプを示す。

## 3 インプリメンテーション

Mach でハンドラを実現するのは、容易である。一つのスレッドを一つハンドラに割り当てることで解決できる。今回の拡張では、スレッドをリアルタイムであるものとそうでないものを区別し、また、スレッドのスケジューリングポリシーをタイムシェアリングと preemptive deadline-driven スケジューリングかを区別することにより実現した。インプリメンテーションの概要は

- イベント通知が必要になると、リアルタイムのスレッドを *cthread_fork*() [8] によって作成し、スケジューリングポリシーを preemptive deadline-driven スケジューリングに変更する、
- 通常の Mach のスレッドはシステムの動作に必要なものを除き、すべてのイベント通知が終了するかウエイトするまで実行されない、
- リアルタイムのスレッドがウエイトしているか存在しないときは、通常のスレッドは、通常の Mach のカーネルがスケジュールする、

である。

イベント通知のために、いくつかのシステムコールを追加した。イベント通知に関係するシステムコールは以下のものである。

- *event_checkin*() — イベント定義の構造体と preemptive deadline-driven スケジューリングのための構造体を登録する。
- *event_wait*() — *event_checkin*() で登録したイベントを待つ。イベントが発生するとユーザモードに返り、イベントの発生から通知されるまでの時間 ($\mu$s 単位) を返す。
- *event_checkout*() — イベント定義の構造体と preemptive deadline-driven スケジューリングのための構造体の登録を消し、通常のスレッドに戻る。

- *event_raise*() — あるクラスのイベントを発生させる。
- *event_procmask*() — ハンドラが実行中にブロックするイベントを変更する。
- *event_sigclass*() — イベント定義の構造体のクラスを変更する。

また、スレッドの同期は C-Threads ライブラリ [8] が提供しているものを使うが、複数のスレッドが共有資源を使う場合には、priority inversion に気をつけなければならない [5, 9]。Preemptive deadline-driven スケジューリングでは、プライオリティは使用しないが、同様のことが起こり得る。このために、二つの関数 *event_begin_critical*() と *event_end_critical*() との間で、一時的にスレッドをイベント駆動型に変更し、さらに重要度を最高に上げることによって、クリティカルリージョンを脱出させることにしている。論理的にきれいな方法はまだ見つけていない。

## 4 Temporal Paging System

Temporal paging system は、巨大なデータ領域を持つマルチメディアアプリケーションのリアルタイム性をサポートすることを目的としている。二次記憶から物理ページにスワップインされる時間の上限は予測できないため、リアルタイム性が保証できない。したがって、予め物理ページに pin-down しておかなければならないが、巨大なデータ領域を pin-down するのは現実的でない場合がある。アドレスと時間の二次元空間でメモリを管理する必要がある。

### 4.1 Temporal

通常のメモリは、一次元の配列である。それぞれの内容を $memory[a](a = 0, 1, 2, \ldots)$ と書く。メモリの重要な特徴は、$memory[a]$ に他の値を代入しない限り変化しないことである。さて、ある時間区間 $T \leq t < T + Q$ でだけ定義されるメモリを temporal といい、$temporal[a, t]$ と書くことにする。このとき、$memory[a] \equiv temporal[a, t](0 \leq t < nQ, n = 0, 1, 2, \ldots)$ である。さらに、表記を簡略化するために、複素数表示する。すなわち、$temporal[z](z = a + it)$ と、以後書くことにする。この表記により、今まで $memory[a]$ で使用してきたインタフェースがそのまま使える。例えば、*tempral_t *malloc(unsigned zsize)* は、アドレスと時間の二次元空間の領域を割り当てることになる。一般のメモリと異なる点は、その時間にならなければアクセスできないことである。したがって、*bcopy(tempral_t z1, tempral_t z2, zlength)* は一般にはメモリフォールトになる。

Temporal に対して初期値を与えることができる。*tcopy*()、*tzero*() などによって、初期値をメモリから copy したり、zero clear したりできる。

現在のハードウェアで、temporal を実現するのは効率的でないので、paging と組み合わせる。

$temporal[z]$ $(M \leq a < M + P, T \leq t < T + Q)$ を tempral page と定義する。$P$ を通常のページングのページサイズにすることにより、tempral paging system はページングを使って実現できる。時間区間毎にページテーブルを用意しなければならないが、差分のみを用意することによってページテーブルの数を少なくすることができる。Tempral paging system では、メモリを時間によって管理できるので、巨大なメモリでも一度にすべてをアクセスしなければ、物理ページに存在しているように見せることができる。

## 5 インプリメンテーションのヒストリ

1990 年 8 月ごろから FM-TOWNS 上の Mach2.5 のリアルタイム化の作業を始めた。12 月ごろまでに、preemptive deadline-driven スケジューリングを実現し、評価を行なった。いくつかの実験によって、preemptive deadline-driven スケジューリングの有効性を確認した。また、高度なアプリケーションも作成し、Multimedia/Realtime 拡張の実用性も確認した [2]。ただし、Mach2.5 は、UNIX がカーネルの中

に組み込まれているので、リアルタイムなスレッドのプルエンプションが不可能なときがあり、問題になることがあった。

1991年3月ごろからMach3.0のリアルタイム化の計画を始めた。マイクロカーネルでは、UNIXのシステムコールを実行中でもリアルタイム性を要求する他のスレッドを優先することができる。この結果、カーネルモードでの遅延が減少する。実際、Mach2.5上のシステムと比較して、イベント通知の時間の最悪値が減少した（表1参照）。表のデータはマルチユーザモードで、1. MIDIハンドラだけのとき、2. MIDIハンドラとスピンループ、3. MIDIハンドラとファイルのコピー、4. MIDIハンドラとコンパイルを組合せたときのMIDIハンドラへのイベント通知の遅延時間である。（）内がMach2.5のときの測定値である。それぞれの測定値は10,000回以上の通知をさせて測定したものである。「遅刻」は通知1000回あたり5ms以上遅れた通知の数である。5msというのは、われわれの経験上の値であり、この程度ならば人間が相手の場合問題がないという参考値に過ぎない。測定に際しては、FM-TOWNS (i386使用、約2MIPS、8MBのメモリ搭載) を使った。

I/Oの割り込み、特にディスクの割り込みがイベント通知の途中（カーネルモード）で起こると、イベント通知が遅れることがあった。この区間では実質的なI/Oの割り込み処理を遅らせることにより、通知の遅延を減少させた。その結果、ハンドラの中でのI/Oは非同期に行なわれ、処理の終了の通知はイベント通知によって行なわれる。

表1: MIDIハンドラへのイベント通知の遅延時間 (ms単位)

| 組合せ(MIDI + ) | 最小 | 最大 | 平均 | 遅刻 |
|---|---|---|---|---|
| NULL | 0.78(0.73) | 1.25(1.71) | 0.83(0.75) | 0(0)/1000 |
| spin loop | 0.79(1.00) | 1.29(3.04) | 0.84(1.20) | 0(0)/1000 |
| file copy | 0.78(1.01) | 5.34(7.98) | 1.78(1.09) | 2(7)/1000 |
| compile | 0.79(0.74) | 8.18(18.47) | 1.24(1.35) | 5(15)/1000 |

## 6 まとめ

Multimedia/Realtime 拡張の概要、特にリアルタイムイベント通知とそのスケジューリング方法であるpreemptive deadline-driven スケジューリングについて述べた。リアルタイムイベント通知の機構はモデルとしては単純であるが、利用する上ではかなり細かい点まで指定しなければならないなどの欠点もある。現在オブジェクト指向のライブラリやハンドラのモニタを作成することを検討している。

Temporal paging systemに関しては、現在の段階（1991年11月）では、設計を終了したが、インプリメンテーションは終了していない。評価に関しては別の機会に譲りたい。

## 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] Jun Nakajima, Masatomo Yazaki, Hitoshi Matumoto, "Multimedia/Realtime Extensions for the Mach Operating System," in *Proceedings of the Summer USENIX Conference*, July, 1991.

[3] IEEE, "Realtime Extension for Portable Operating Systems," P1003.4/Draft10, February, 1991.

[4] D. Golub, R. Dean, A. Forin, and R. Rashid, "Unix as an Application Program," in *Proceedings of the Summer USENIX Conference*, June, 1990.

[5] Hideyuki Tokuda, Tatsuo Nakajima, and Prithvi Rao, "Real-Time Mach: Towards a Predictable Real-Time System," in *Proceedings of USENIX Mach Workshop*, October, 1990.

[6] John A. Stankovic and Krithi Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Operating Systems," *ACM Operating Systems Review*, Vol.23, No.3, July, 1989.

[7] C. L. Lui and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, Vol.20, No.1, 1973.

[8] Eric C. Cooper and Richard P. Draves. "C Threads", Technical Report, Computer Science Department, Carnegie Mellon University, CMU-CS-88-154, March, 1987.

[9] Sha, L. and Rajkumar, R., and Lehoczky, J.P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," Carnegie Mellon University, CMU-CS-87-181, 1987.

[10] J. S. Sventek, "An Architecture supporting Multi-media Integration," *IEEE Computer Society Office Automation Symposium*, Gaithersburg, MD, April, 1987.

[11] David L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System," *IEEE Computer*, Vol.23, No.5, 1990.