システムソフトウェアと
オペレーティング・システム 67－2
(1994. 12. 8)

OSレベルでのリソース・リザベーションのメディア並列処理に対する効果

松尾聡* 岡村耕二 荒木啓二郎 福田晃

奈良先端科学技術大学院大学 情報科学研究科
*ミノルタ（株） 西神情報センター

〒630-01 奈良県生駒市高山町8916-5 奈良先端科学技術大学院大学 情報科学研究科
〒651-22 神戸市西区高塚台4-4-1 ミノルタ(株) 西神情報センター

あらまし 我々は、メディア処理の時間的制約を保証するために、オペレーティングシステムのスケジューリング方式や資源管理方式に時間的制約の保証を行う機能を組み込む研究を行っている。メディア処理において、特に複数のメディア処理を同時に行う場合、それらのメディア処理の個々の時間的制約を保証することを考慮した資源予約は重要である。本稿では、メディア処理の時間的制約を保証するためのリソース・リザベーションの方式をCPUに着目した場合について、その方式とメディア処理のスケジューリング方式を示し、その実装結果について考察を行う。

和文キーワード リソース・リザベーション, メディア処理, スケジューリング, 時間的制約, 実時間処理, オペレーティングシステム

The effect of resource reservation for parallel media processing at operating system level

Satoshi Matsuo*, Koji Okamura, Keijiro Araki and Akira Fukuda

Graduate School of Information Science, Nara Institute of Science and Technology
*Minolta Co.,Ltd. Advanced System Center of Seishin

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma 630-01, JAPAN
*Minolta Co.,Ltd. Advanced System Center of Seishin
4-4-1 Takatsukadai, Nishi-ku, Kobe 651-22, JAPAN

Abstract

We have been researching the mechanisms which guarantee timing constraint for media processing in scheduling and resource management at operating system level. In the case of the multiple media processing, it is important to guarantee timing constraint of each medium processing. We propose the mechanism of the resource reservation for medium processing which needs to guarantee the timing constraint in this paper. We specially show the reservation algorithm of CPU and the scheduling of the medium processing, and evaluate the algorithm through experiments.

英文 key words resource reservation,medium processing,scheduling,timing constraint,real-time processing,operating system

1 はじめに

近年、CPUの高速化やディスクの大容量化が飛躍的に進み、コンピュータ上でのメディア処理に対する期待が大きくなってきている。

メディア処理は従来の数値計算等の処理とは異なり、その処理に対して時間的制約を持つという特徴がある。例えば動画を再生する処理を考えた場合、各フレームを録画した時の時間軸にあわせて再生を行う事が必要である。この場合、時間軸に従った再生を行うことが再生時の時間的制約になる。この時間的制約を保証することが、メディア処理を保証することになる。

ところで、メディア処理はデッドラインを持った周期的な処理を扱う実時間処理とその性質において非常に似ている。しかし、実時間処理は、それぞれのプロセスのデッドラインをプライオリティの高いプロセスから保証することを念頭においているため、プロセス間でのCPUの細かい配分までは規定していない。一方、マルチメディア処理ではその処理によって再生されたメディアの品質も考慮する必要があるため、複数プロセス間でCPUを予約確保させる時にその配分方法までも指定できた方が良い。現在、存在している実時間処理用オペレーティングシステムはこのような機構が不十分なため、メディア処理を行うことには適していない。そこで、本研究では2種類のCPUの予約確保方式(周期指向、割合指向)を提案する。

我々は、メディア処理に必要とされる計算機資源、例えばCPU、メモリ、ネットワーク等をリソースと呼び、メディア処理を保証するためにリソースを予約確保し処理が終了するまで保証することをリザベーションと定義し、リソースのうちCPUのリザベーションのモデルについて提案した[1]。本稿では、リザベーションモデルを実装するためのアルゴリズムの提案と、そのアルゴリズムを検証した結果について述べる。

以下、第2章ではメディア処理で保証すべきサービスの品質について述べ、第3章では既存の実時間プロトコルをメディア処理に適用した場合の問題点について述べる。第4章では本稿で提案するメディア処理用スケジューリングについて述べ、第5章で本稿で提案するアルゴリズムの評価と考察を行い、第6章で今後の課題とまとめを述べる。

2 メディア処理

本章では、メディア処理についてその特徴とシステムがメディア処理を行うために保証すべきサービスの品質(Quality of Service、以後、QOS)について述べる。

2.1 メディア処理の特徴

メディア処理の特徴は、その処理に時間的制約があることである。例えば、1秒間に30フレームで動画を再生する処理を考える。動画の1フレームを描画するのに必要な処理時間を$N(msec)$とすると$1/30sec$の間に$N(msec)$をCPUに割り当てることを保証すれば、30フレーム$/sec$で動画を再生することができる(図1)。

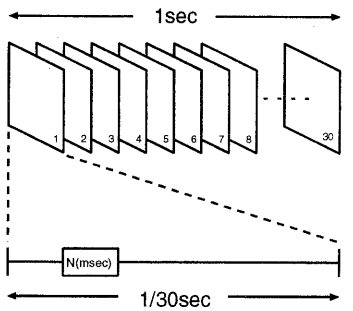

図1: 30フレーム$/sec$での動画再生処理

ここで、30フレーム$/sec$は時間的解像度、1フレームの処理時間$N(msec)$を空間的解像度に対応する。

2.2 メディア処理のためのQOS

時間的解像度と空間的解像度を指定することでメディア処理におけるQOSは指定できる[2]。QOSが保証されると、そのメディアは滑らかに再生され、受け手にも不快感を与えることがない。しかし、現在のオペレーティングシステムではQOSの保証ための機構が不十分なためにQOSを完全に保証することができない。具体的には、CPUの使用状況は動的に変化するために、要求された処理に対してCPUを周期的に割り当てることができず、時間的解像度を保証することができないためである。そのために、メディア処理に対してCPUを予約確保し、それを保証する必要がある。

そこで、我々はCPUの予約確保を行ない、その予約確保されたCPUを周期的に処理に割り当てることを保証するリザベーション方式を提案する。

3 既存の実時間プロトコル

本章では、既存の実時間プロトコルとそのプロトコルのメディア処理時の問題点について述べる。

実時間プロトコルは実時間処理を行うためのものである。実時間処理と性質が似ているメディア処理に実時間プロトコルを適用する場合を考える。

3.1 実時間プロトコル

実時間処理のためにいくつかの実時間プロトコルが提案されている[3]。実時間プロトコルは複数処理によるリソースの競合問題に対して用いられる。以下に実時間プロトコルのうちの代表的なものを示す。

Priority Ceiling Protocol (PCP)

PCPは相互デッドロックを解決する。クリティカルセクションに入ったプロセスにその処理が終るまで一時的に全てのプロセスのプライオリティより高い値(シーリング値)のプライオリティを与え、他のプロセスの割り込みを排他制御する。最高値を割り当てられたプロセスのプライオリティはクリティカルセクションをぬけると、元のプライオリティに戻る。

Priority Inheritance Protocol (PIP)

PIPは優先度逆転(Priority Inversion)を解決する。低いプライオリティの処理がCPUのクリティカルセクションに入った時に、その処理よりプライオリティの高い処理がCPUをプリエンプションしクリティカルセクションに入ろうとすると、先に入った処理のプライオリティにその時点でのプライオリティの最高値を与えてやりクリティカルセクションの処理を終えさせる。最高値を割り当てられたプロセスのプライオリティはクリティカルセクションをぬけると、元のプライオリティに戻る。

3.2 メディア処理時の問題点

これらの実時間プロトコルでメディア処理を行う場合を考える。メディア処理はプライオリティとは関係のない周期時間を持っている。実時間プロトコルでは、最もプライオリティの高い順にそのデッドラインを保証することを目標にしているため、複数プロセス間のCPUの配分方法は考慮されていない。そのため、お互いのプライオリティの存在していないメディア処理では、個々のメディアのQOSを保証できない。この理由により、実時間プロトコルをメディア処理に用いることは適していない。

そこで本稿では、実時間プロトコルに代わる新たなメディア処理用スケジューリングを提案する。

4 メディア処理用スケジューリング

本章では、メディア処理の時間的制約の要求内容について述べ、我々の提案する2種類の資源予約のためのスケジューリング方法とその動作例についての説明を行う。

4.1 資源予約方法

メディア処理の時間的制約の要求は、「ある処理はCPUの30%、別の処理はCPUの20%が使いたい」という形で一般的に出される[4]。この要求の場合、30%と20%はそれぞれCPUの割合を指定している点は同じであるが、それぞれの数字が持っている意味は異なっている場合がある。それは指定された周期時間内の処理時間の扱い方が異なるためである。扱い方は2つの場合が考えられる。1つめは同期を考慮して複数の処理を行う場合で、それぞれの処理時間を分割して交互に処理することが期待される。2つめの同期を考慮しない処理の場合、処理時間を可能な限り一括して行うことが期待される。そこで、これらの2つの扱い方を満たす複数メディアのためのCPUのリザベーション方法を提案する。それは、周期時間内の処理時間をできる限り一括して行う周期指向型資源予約方式と周期時間内の処理時間を複数に分割し同期を考慮して行う割合指向型資源予約方法である。

我々の提案する資源予約方法を以下に示す。

周期指向型資源予約

周期指向型資源予約(Period Oriented Reservation)は、デッドラインの最も近い処理を優先的にCPUに割り当てる。一回のCPUの割り当てはできる限り長い時間を与え、周期時間内の処理をできるだけ分割せずに行うことを目的としている。

割合指向型資源予約

割合指向型資源予約(Rate Oriented Reservation)は、周期時間のデッドラインまでの時間に対する残りの処理時間の割合の大きいものを優先的にCPUに割り当てる。このアルゴリズムは、全体の処理を均等に行うことを目的としている。

4.2 実装方法

我々の提案する資源予約を実現するために、拡張した機構について説明する。

- メディア処理プロセス用レディキュー
  通常のプロセスのレディキューとは別に、メディア処理プロセス用のレディキューを拡張、メディア処理用レディキューから優先的にプロセスの選択を行う
- メディア処理プロセス管理用データ
  - 要求周期時間 (*period_order*)
    処理時間を終えるための最大時間間隔を指定した時間
  - 要求処理時間 (*job_order*)
    周期時間内で行う処理時間を指定した時間
  - 実周期時間 (*period_time*)
    プロセス管理用に用いる周期時間
  - 実処理時間 (*job_time*)
    プロセス管理用に用いる処理時間

また、どちらのアルゴリズムも予めメディア処理用のプロセスを資源予約を行う必要がある。以下の動作によりメディア処理用プロセスの資源予約を行われる。資源予約されたプロセスは、その処理が終了するまでリソースを保証される。

1. まず、資源予約のため基準単位時間(以後、*unit_time*)を決める。資源予約の*unit_time*は、コンテキストスイッチが起こってから次のコンテキストスイッチが起こるまでの時間とする。ここでは、CPUのコンテキストスイッチは*unit_time*を越えて起こることはないとする。

2. 次に、メディア処理を行うプロセスの資源予約を行う。資源予約は *unit_time* に基づいた要求周期時間 (以後、*period_order*)、要求処理時間 (以後、*job_order*) の 2 つの値で行う。*period_order* は処理を行うための周期時間、*job_order* は周期時間内で行う処理時間で、*unit_time* の整数倍の値で指定する。

3. 資源予約の要求を受けると、その時点で資源予約されているメディア処理用プロセスの $\sum(job\_order/period\_order)$ を計算し、その値に要求されているプロセスの $job\_order/period\_order$ を加えた結果が CPU の処理能力を越えなければ、その要求は受理され資源予約が行われる。越えた場合には、要求元に受理不可を通知し、資源予約は行われない。

4. 資源予約処理では、要求されたプロセスをメディア処理用レディキューに登録する。次に、実処理時間 (以後、*job_time*) に要求処理時間 (以後、*job_order*)、*period_time* に *period_order* の値をコピーする。

5. 2〜4 を繰り返し行う。

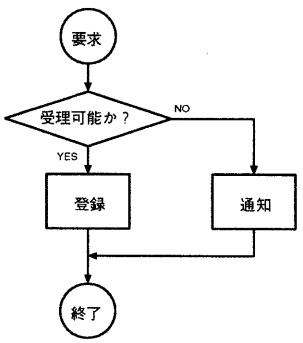

図 2: 資源予約の流れ

割合指向型資源予約におけるプロセス管理は以下の手順で行われる。

1. タイマー割り込みにより CPU のコンテキストスイッチが起こると、メディア処理用レディキューを検索する。キューに登録されているプロセスが 0 の場合、通常のレディキューを検索しにいく。

2. キューにプロセスがある場合、プロセスの *job_time*/*period_time* の値を計算し、その値が最大 (> 0) のプロセスを選択し、選択したプロセスの *job_time* から *unit_time* を減算する。最大値を持つプロセスが複数ある場合、*period_time* が最小のプロセスを選択する。

3. メディア処理用レディキューの全てのプロセスの *period_time* から *unit_time* を減算する。*period_time* $= 0$ の場合、*period_time* に *period_order*、*job_time* に *job_order* の値をコピーする。

4. 選択したプロセスを実行する。メディア処理用レディキューでプロセスを選択できなかった場合、すなわち全てのプロセスの *job_time* $= 0$ の場合、プロセスはいずれも次の周期待ちの状態でありメディア処理用プロセスは実行せず、通常のレディキューに検索にいく。

5. 2〜4 を繰り返し行う。

周期指向型資源予約におけるプロセス管理は、Deadline Driven Scheduling Algorithm[5] を適用することができる。

4.3 動作例

先に述べた 2 種類の資源予約が同じ条件の処理を行うときの動作例を示す。

条件

CPU の基本単位時間を 1 とし、周期時間 10 における処理時間 2 の処理 A、周期時間 8 における処理時間 3 の処理 B の 2 つを同時に資源予約する

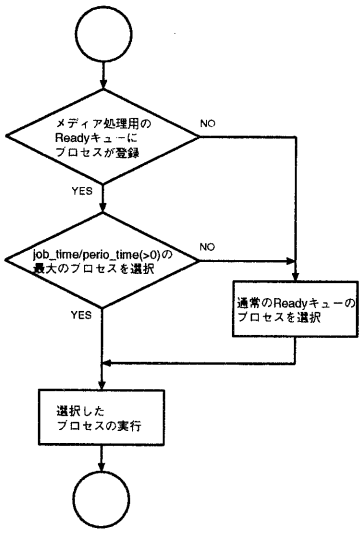

図 3: プロセス管理の流れ

表 1: 各単位時間における状態と選択結果

	処理 A		処理 B		選択した処理
	DL	JT	DL	JT	
0	10	2	8	3	処理 B
1	9	2	7	2	処理 B
2	8	2	6	1	処理 B
3	7	2	5	0	処理 A
4	6	1	4	0	処理 A
5	5	0	3	0	×
6	4	0	2	0	×
7	3	0	1	0	×
8	2	0	8	3	処理 B
9	1	0	7	2	処理 B
10	10	2	6	1	処理 B
11	9	2	5	0	処理 A

4.3.1 周期指向型資源予約

表 1はそれぞれの処理のデッドラインまでの時間 (DL)、デッドラインまでに残されている処理時間 (JT)、選択した処理を単位時間ごとに表し、図 4は、実行結果を示したものである。表 1の×は処理 A、B のいずれも選択しないことを表す。

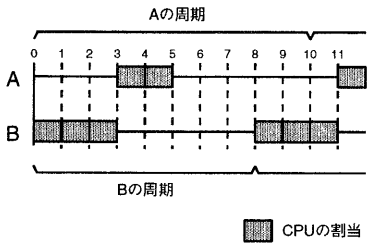

図 4: 周期指向型資源予約の実行例

4.3.2 割合指向型資源予約

表 2はそれぞれの処理のデッドラインまでの時間 (DL)、デッドラインまでに残されている処理時間 (JT)、選択した処理を単位時間ごとに表し、図 5は、実行結果を示したものである。表 1の×は処理 A、B のいずれも選択しないことを表す。

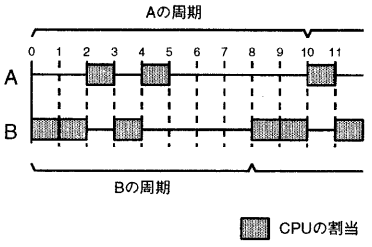

図 5: 割合指向型資源予約の実行例

表 2: 各単位時間における状態と選択結果

	処理 A		処理 B		選択した処理
	DL	JT	DL	JT	
0	10	2	8	3	処理 B
1	9	2	7	2	処理 B
2	8	2	6	1	処理 A
3	7	1	5	1	処理 B
4	6	1	4	0	処理 A
5	5	0	3	0	×
6	4	0	2	0	×
7	3	0	1	0	×
8	2	0	8	3	処理 B
9	1	0	7	2	処理 B
10	10	2	6	1	処理 A
11	9	1	5	1	処理 B

5 評価と考察

本章では、先に述べた割合指向型スケジューリングの評価実験とその考察について述べる。

5.1 評価実験内容

今回行なった実験は、以下の環境で行なわれた。

- DECstation 3000/300
- Portable Thread Library (PTL)[6]

割合指向型スケジューリングは、PTL のスケジューラを拡張することで実現した。PTL 上で実装を行ったのは、PTL がマルチスレッド環境でオリジナルのプロトコルの実装が容易で、検証を行い易いためである。

5.2 実験結果と考察

今回の実験では、メディア処理を単一で行う場合と複数のメディア処理を同時に行う場合の 2 通りについて測定した。スケジューリングの検証が目的であったために、どちらの場合も他の負荷をかけない状態で測定を行った。基準単位時間は $100msec$ である。

5.2.1 単一のメディア処理の場合

ここでは、異なる周期時間と処理時間を指定した 6 つのメディア処理を実行し測定を行った。測定結果はそれぞれ 100 回ずつ測定した値の平均値である (表 3)。

表 3: メディアの単一処理結果

	指定値		測定値	
	周期時間	処理時間	周期時間	処理時間
A	500	100	500.0	101.6
B	1000	100	999.2	100.6
C	2000	100	1999.7	98.6
D	500	200	499.6	201.1
E	1000	200	999.7	200.9
F	2000	200	1999.6	200.5

(単位は $msec$)

5.2.2 複数のメディア処理の場合

ここでは、異なる周期時間と処理時間の複数の処理を同時に実行し測定を行った。測定結果はそれぞれ 100 回ずつ測定した値の平均値である (表 4)。

5.2.3 考察

今回の測定結果は他の負荷をかけない状態のものであるため、理想に近い値を得ることができた。また、PTL 上での実装のため、ユーザレベルでの実行結果になっている。実際には、カーネルモードでの実行になるため、カーネルモードで発生する特有の問題について考える必要がある。

表 4: 複数のメディア処理結果

	指定値		測定値	
	周期時間	処理時間	周期時間	処理時間
A	500	100	499.5	100.0
B	1000	100	1000.2	100.6
A	500	100	499.7	100.3
E	1000	200	1001.6	199.8
A	500	100	499.7	101.4
B	1000	100	999.9	98.7
C	2000	100	2000.1	100.3

(単位は *msec*)

6 今後の課題とまとめ

本稿では、時間的制約のあるメディア処理を行うためのスケジューリングについて述べた。今回は、そのうちの割合指向型スケジューリングについて実装、検証を行った。評価によってスケジューリングはメディア処理の時間的制約を保障することに有効であるという結果が得られた。課題として、基準単位時間の設定に対するオーバヘッドの考慮があり検討をすすめる。

また、我々はアプリケーションの様々な要求に対応するために周期指向型スケジューリングと割合指向型スケジューリングの併用を検討している。そのためには、2 つのスケジューリングを調停する必要があり、そのための問題点を解決しなければならない。

本稿では PTL 上での実装であったが、今後はオペレーティングシステム上に 2 つのスケジューリングの実装し、それによって発生する問題点について検討し改良を行い、CPU のリザベーションの完成を目指す。さらに、他のリソースのリザベーションについても研究を行う予定である。

謝辞

本稿の執筆に際して御助言を頂いた奈良先端科学技術大学院大学の荒木研究室、福田研究室の皆様に感謝致します。

参考文献

[1] 松尾, 岡村, 荒木, 福田, “マルチメディア処理における OS レベルでのリソース・リザベーション”, 情報処理学会 システムソフトウェアとオペレーティング・システム研究会, 63-10, Mar. 1994.

[2] 岡村, 稲垣, 松尾, 荒木, 福田, “マルチメディア同期機構の試作と評価”, 情報処理学会 マルチメディアと分散処理, DPS-65-14, Mar. 1994.

[3] L.Sha, R.Rajkumar, S.Son, and C.Chang, “A Real-Time Locking Protocol”, IEEE Transactions on Computers, 40(7), pp.793-800, Jul. 1991.

[4] 藤田, 中島, 手塚, “連続メディア処理のためのプロセッサ管理方式, 日本ソフトウェア科学会, B6-3, Oct. 1994.

[5] C.L.Liu, James W.Layland, “Scheduling Algorithm for Multiprogramming in a Hard-Real-Time Exvironment”, Journal of the Association for Computing Machinery, Vol.20, No.1, pp.44-61, Jan. 1973.

[6] 安倍, 松浦, 谷口, “BSD UNIX の下でのポータブルマルチスレッドライブラリ PTL の実現”, 情報処理学会研究報告, 情報, 1994.