システムソフトウェアと
オペレーティング・システム
75－9
（1997．6．5）

UNIX 上で周期スレッドを実現する ユーザレベルスレッドライブラリの実現法

安倍広多[†], 松浦敏雄[†], 安本慶一[††], 東野輝夫[†††], 谷口健一[†††]

[†] 大阪市立大学 学術情報総合センター
[††] 滋賀大学 経済学部
[†††] 大阪大学大学院 基礎工学研究科

あらまし

動画像や音声などの実時間制約を伴う情報を扱うマルチメディアシステムの実現のために, 時間制約を満たしながら周期的に動作するマルチスレッド機構が注目されている. このようなスレッド機構には, リアルタイム OS で実現されるものが存在するが, 実時間制約を保証できない OS(UNIX) のもとでの実現方法は明らかではなかった. 本論文では, UNIX のもとで時間制約を満たしながら周期的に動作するマルチスレッドライブラリ RPTL(Real-time Portable Thread Library) の設計と実装方法について述べる. 本稿で提案する方法は, RPTL が観測したスレッドの動作状況を表すヒントをもとに, RPTL とアプリケーションが協調して可能な限り時間制約を満たすように努めるものである. RPTL のプロトタイプを実装し, 実験によりその有効性を示す.

キーワード　実時間システム, ソフトリアルタイム, スレッドライブラリ, QoS, UNIX

Design and Implementation of a Soft Real-time Thread on UNIX

Kota ABE[†], Toshio MATSUURA[†], Keiichi YASUMOTO[††], Teruo HIGASHINO[†††], Kenichi TANIGUCHI[†††]

[†] Media Center, Osaka City University
[††] Faculty of Economics, Shiga University
[†††] Graduate School of Engineering Science, Osaka University

Abstract

In order to create multimedia systems which handle continuous media such as video and audio data, real-time multi-thread mechanisms are often used. Such mechanisms are usually constructed on a real-time kernel. However, it is not well-known how to realize such mechanisms under an OS which cannot ensure a timing constraint.

In this paper, the design and implementation of a multi-thread library RPTL(Real-time Portable Thread Library) on UNIX is demonstrated. RPTL cooperates with applications to satisfy timing constraints by providing some appropriate hints. We discuss the implementation of the prototype library and also present some experimental results.

Keywords　Real-time System, Soft Real-time, Thread Library, QoS, UNIX

1 はじめに

動画像や音声などの時間制約をもつデータを取り扱うマルチメディアシステムを実現するには, 周期的に一定量の CPU を割り付けることが重要である. このために, 時間制約を OS レベルで扱うことができるリアルタイム OS(RTOS) 上で, マルチメディアシステムを実現する方法が研究されている [6]. しかし, 既存の多くのシステムでは, UNIX 等の非リアルタイム OS が用いられており, したがって, この上でマルチメディアシステムが動作すれば, 多くの利用者によって有益であると考えられる.

非リアルタイム OS では, 他のプロセスの活動などによって, プロセスが使用できる CPU 資源が動的に変化するので, 周期的に一定量の CPU を割り付けることができない場合が生じうる. このような場合, 利用可能な CPU 資源の量に応じて, アプリケーション自身が, 動画像や音声等のサービス品質 (QoS) を動的に制御する仕組みが必要となる.

本稿では, UNIX 上で, 利用可能な CPU 資源に適応したスケジューリングを行う一手法を提案する. さらに, このようなスケジューリングを可能にするマルチスレッドライブラリ RPTL(Real-time Portable Thread Library) のプロトタイプを作成し, 実験によりその有効性を示す.

2 設計方針

RPTL を実装する上で, 以下のような設計方針を定めた.

2.1 アプリケーションとの協調

非リアルタイム OS で実装する以上, デッドラインミスは避けられない. マルチメディアシステムではデッドラインミスは致命的ではないが, 画像の揺らぎや音飛び等が発生することがある.

このため, デッドラインミスしないように, RPTL とアプリケーションが協調した動作を目指す. すなわち, RPTL 側でアプリケーションにスケジューリングヒントを提供し, それによってアプリケーションが QoS を制御することによって可能な限りデッドラインミスを回避する.

2.2 汎用性

なるべく一般的なシステムで動作することを目指す. 具体的には以下の方針をとる.

- 既存環境との親和性を考え, システムで実行中の他のプロセスについては特に前提を設けない. 特別な QoS サーバや資源サーバが全プロセスを統括したりすることはない.
- アプリケーションがさまざまな計算機環境で動作することを考慮し, 各周期で必要な計算時間等はあらかじめ与えない.

3 UNIX のもとでの実現上の問題点

前述の方針を満たす既存のスケジューリング方式で, UNIX でも利用可能と考えられる方式に, Real-Time Mach 上で実装されたセルフスタビライズ法 [2](以下 SS 法と略す) がある. これは,

- デッドラインミス数が規定値に達すると, 周期スレッド自身が自分の周期を延ばす (QoS を低下させる),
- デッドラインミスが生じない場合, 周期を縮める (QoS を増加させる).

というアルゴリズムによって, システムの負荷に応じて自律的に周期を調節する方法である.

しかし, SS 法を UNIX 上で再実装し実験した結果, 以下のような問題があることが分かった.

- 他のプロセスが活動していない時は, 指定する周期, 処理量によっては CPU をほとんど占有しようとする. このため, 他のプロセスの活動によってデッドラインミスしやすい. (6.2節参照)
- アプリケーションの CPU 要求が高く, 実行中の環境では要求した周期で実行できない場合, デッドラインミスを生じない周期まで一旦下がるが, その状態から常に周期を縮めようとするため, 周期が安定しない. (6.3節参照)

スケジューリング方法以外にも, I/O やページングによるプロセスのブロッキングの問題や, ファイルシステムが連続メディアに適していない問題等があるが, 本稿では触れない.

4 提案する方式

著者らは非リアルタイムシステム (UNIX) 上でマルチメディア処理を行うのに適したスケジューリング方式として, セルフスタビライズ法を拡張した CPU 資源モニタ法を提案する.

4.1 スケジューリングヒントの収集 (CPU 資源の測定)

適当な測定期間 T_P 中, どのスレッドも実行中でない時間をアイドル時間 T_I とし, プロセスが消費した CPU 時間 (プロセスのために消費したシステム時間, ユーザ時間の合計. getrusage システムコールで得られる) を T_C とする. プロセスアクティビティ, CPU 利用率を以下のように定義する.

$$\text{プロセスアクティビティ} = 1 - \frac{T_I}{T_P}$$

$$CPU\ \text{使用率} = \frac{T_C}{T_P - T_I}$$

直観的には, プロセスアクティビティは, プロセスが測定期間のうち, どの程度の時間 CPU を使用したいかを表す. 測定期間中, 完全にビジーだった場合は 1.0 になる. CPU 使用率は, プロセスがアイドル状態でなかった時間のうち, どれだけ CPU を使用できたかを表す. 他のプロセスの活動によって CPU が横取りされなければ理想的には 1.0 になる.

4.2 アプリケーション側のスケジューリング処理

スレッドがデッドラインをミスするということは, 現在の CPU 使用率ではスケジューリングできない計算量をかかえていると判断する. CPU 使用率を上げることはできないため, プロセスアクティビティを減少させることによって対処する. また, プロセスアクティビティが高い状態では, 他のプロセスによって CPU を奪われるとデッドラインをミスしやすいため, プロセスアクティビティを一定以下に抑える制御を行う.

システムは定期的にプロセスアクティビティ, CPU 使用率およびデッドラインミスの回数を計測し, スケジューリングヒントとしてアプリケーションへ提供する.

以下に, アプリケーション側の制御例を示す.

- 周期スレッドが, デッドラインを連続してミスした, あるいはプロセスアクティビティが一定値を越えた場合, 自分自身の周期を延ばし, プロセスアクティビティを減少させる. (動画像の表示などの場合, 空間的解像度を減少させるなどの方法によって周期当たりの計算量を減らしても良い)
- 「デッドラインをミスせず, プロセスアクティビティが一定値以下, CPU 占有率が一定値以上である状態」を連続して達成した場合, 周期を縮める. 縮めたら, プロセスアクティビティが変化するまで縮めない.

4.3 CPU 資源モニタ法の利点

この方法の利点を以下に示す.

- プロセスアクティビティを参照することによって, プロセスがどの程度 CPU を占有しても良いかを自由に決定できる. アクティビティを抑えることによって, 他のプロセスの活動開始による周期変動を軽減することができる.
- アプリケーションの CPU 要求が高く, 実行中の環境では要求した周期で実行できない場合でも, プロセスアクティビティを参照することによって周期をスケジュール可能な範囲に留めることができる.
- 過負荷状態から脱した時に, CPU 占有率, プロセスアクティビティに注意しながらスレッドの周期を縮めるので, 周期を比較的安定に保つことができる.
- 必要な計測コストが低い. T_Iはプロセスがアイドル状態になった時刻およびアイドル状態から脱した時刻を計測すれば求められる. T_Cを得るには getrusage システムコールを発行すれば良い.

5 実装

RPTLの実装(プロトタイプ)は, 周期スレッドを実行できるようにPTL(Portable Thread Library)[1] を拡張し, その上でCPU資源の測定機能を追加することによって行った. RPTLはPTLの移植性を受け継いでいるため, 一般的なBSD UNIX上ならば動作する.

スレッドの周期的スケジューリング, デッドライン判定, タイムスライスによるプリエンプション等のために, setitimer システムコールによるインターバルタイマ割り込みを用いている.

5.1 API

PTL では既に POSIX1003.4a(Draft6) ベースの API(Pthreads[7]) が実装済である. これに, 以下の周期スレッドサポート用の API を追加した.

pthread_rt_start()

実行中のスレッドのコンテキストで, 指定した周期関数を周期的に呼び出す. この呼び出しは, 周期関数中で pthread_rt_exit() を呼び出すまで続く.

pthread_rt_exit()

周期関数の実行を終了させ, pthread_rt_start() から復帰させる.

pthread_rt_deadline_handler()

本 API を呼び出した関数は他のスレッドがデッドラインをミスするまでブロックする. デッドラインをミスしたスレッドはサスペンドされ, 本APIが復帰する.

pthread_rt_deadline_miss()

自スレッドがデッドラインをミスしているかどうかを返す.

pthread_rt_resume()

デッドラインミスでサスペンドされたスレッドを再開する.

5.2 CPU資源の測定法

PTLのアイドルスレッドは, select システムコールによってプロセスをブロックさせていた. RPTLでは select システムコールの直前に gettimeofday システムコールを発行し, アイドル状態となった時刻を記憶するように修正した.

アイドル状態から起床するのはシグナル割り込みか, 入出力可能となって select がリターンした場合である. この時にも gettimeofday を発行し, アイドル時間の総計を算出する.

RPTLは測定期間毎に使用率測定関数を呼び出す. この関数では getrusage システムコールによって, 前回関数が呼ばれた時からのプロセス仮想時間を計算する.

測定期間は, 長すぎると現在の状況を正しく測定できず, 短すぎると測定誤差が大きくなる. 今回は3秒とした.

測定データはアプリケーション側で自由に参照できる.

6 実験結果

実装した RPTL のプロトタイプを用いて, 以下の実験を行った. 計算機環境は PC-AT 互換機 (AMD-5x86 133MHz, Memory 48MB, NetBSD-1.1) を使用した.

6.1 RPTLの周期性測定

他のプロセスの影響がない状態で, RPTLがどの程度, 周期を正確に実行できるかを調べた. 10msec, 20msec の周期をもつ二つの周期スレッドを生成し, 周期関数の中で前回の周期からの時間を測定した. 10msec は RPTL の最小のスケジューリング間隔であり, UNIX のインターバルタイマの粒度からきている.

測定結果を図1に示す. これから, ほぼ正確な周期で実行されていることがわかる.

6.2 他のプロセスによる影響

RPTL上で SS 法および CPU 資源モニタ法を実装し, それぞれ他のプロセスの影響を受けた場合のスケジューリング動作を調べた.

SS 法でのスケジューリングは, 3回連続してデッドラインミスした場合周期を 10msec 延ば

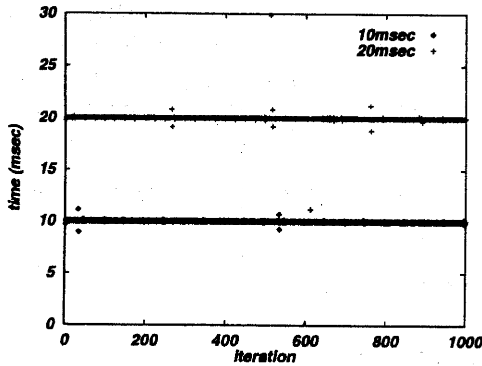

図 1: RPTL の周期実行の測定結果

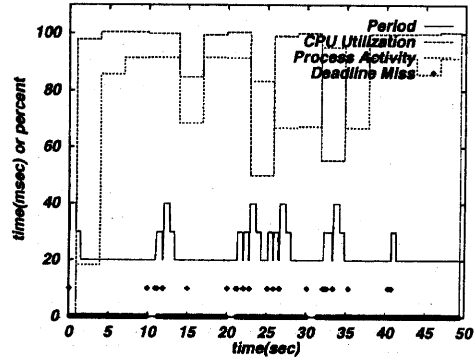

図 2: 他プロセスの影響下でのセルフスタビライズ法

し, 20 回連続してデッドラインをミスしなかった場合に周期を 10msec 縮めるようにして行った.

CPU 資源モニタ法では以下のポリシーを用いた. (4.2 節参照)

- 周期スレッドが, デッドラインを連続して 3 回ミスした, あるいはプロセスアクティビティが 0.75 を越えた場合, 周期を 10msec 延ばす.
- 「デッドラインをミスせず, プロセスアクティビティが 0.6 以下, CPU 占有率が 0.8 以上である状態」を連続 20 回継続した場合, 周期を 10msec 縮める. 縮めたら, プロセスアクティビティが変化するまで縮めない.

周期スレッドを実行するプロセス P1 の実行開始 10 秒後に, 空ループを 5 秒間実行するプロセス P2 を 5 秒おきに 3 回実行した. P1 では 18msec を要する処理を 20msec の周期で実行させた. 周期, CPU 利用率, プロセスアクティビティ, デッドラインミス (デッドラインをミスした場合 $y = 10$, ミスしなかった場合 $y = 0$ にプロット) の様子を 2, 3 に示す.

SS 法では, 周期が細かく変動し, CPU 資源モニタ法ではゆっくりと変化することがわかる. CPU 資源モニタ法の方が, 長い周期へ収束しているが, これはプロセスアクティビティを一定以下に抑える機能が働いているためである.

$y = 0, y = 10$ のどちらにもプロットされていない個所は, その期間 CPU が割り当てられなかったことを意味するが, SS 法では P2 起動

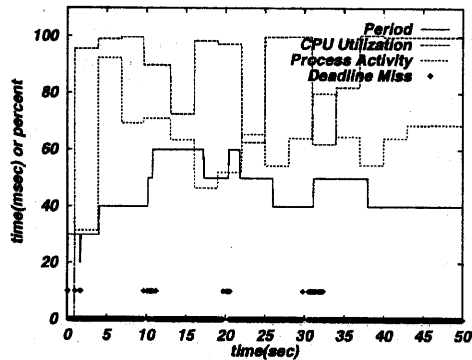

図 3: 他プロセスの影響下での CPU 資源モニタ法

時にこのような個所が多数発生しているのに比べ, CPU 資源モニタ法では比較的 CPU を割り当てられている. これは CPU 資源モニタ法の方が余裕をもったスケジューリングをしているためと考えている.

6.3 CPU 要求が高すぎる場合の動作

37msec を要する処理を 10msec の周期で実行させようとした場合の結果を図 4, 図 5 に示す.

SS 法では周期を縮めようとして周期が変動するのに対し, CPU 資源モニタ法では, 安定した周期が得られることがわかる.

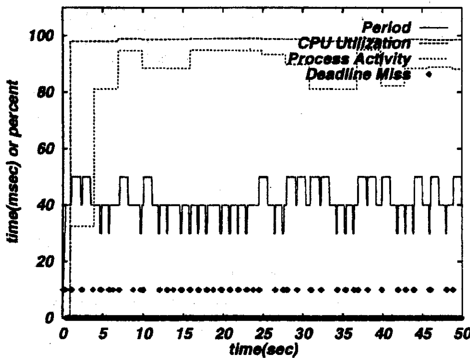

図 4: CPU 要求が高すぎる場合のセルフスタビライズ法

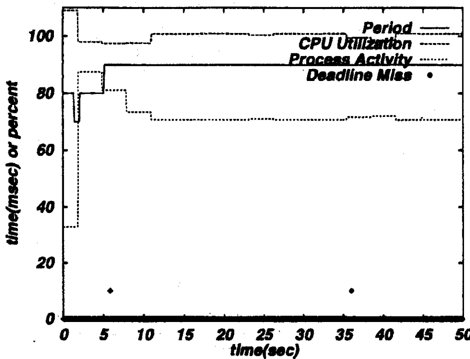

図 5: CPU 要求が高すぎる場合の CPU 資源モニタ法

7 むすび

本稿では, UNIX 上で周期スレッドを実現するためのスケジューリング方法として, アプリケーションと協調して動作する CPU 資源モニタ法を提案, 実装し, その有効性を確認した.

今後の課題としては,

- 測定値 (プロセスアクティビティ, CPU 使用率) の高精度化
- セルフスタビライズ法以外のアルゴリズムとの比較
- 最適なスケジューリングパラメータの検討 (パラメータの自動算出方法の検討を含む)
- より優れたアプリケーション側のスケジューリング方法の検討 (例えば, 過去のプロセスアクティビティ, CPU 利用率, デッドラインミス率などを記憶し, 将来のスケジューリングに役立てるなど)
- 実際のアプリケーションに適用, 有用性の調査

等があげられる.

参考文献

[1] 安倍広多, 松浦敏雄, 谷口健一: BSD UNIX 上での移植性に優れた軽量プロセス機構の実現, 情報処理学会論文誌, Vol.36, No.2, pp.296–303, (1995-02).

[2] 舩渡大地, 徳田英幸: Real-Time Mach 3.0 における連続メディアサーバの実験, 情処研報 93-OS-60, pp.75-82. (1993).

[3] 徳田英幸: Real-Time Mach: 実時間マイクロカーネル, 情報処理, Vol.37, No.12., pp.1117–1124 (1996).

[4] Real-Time Mach Project, School of Computer Science, Carnegie Mellon University: *Real-Time Mach 3.0 User Reference Manual,* (1995).

[5] 河内谷, 他: 連続メディア処理のためのリアルタイムスレッドモデルの拡張, 第 48 回情処全大論文集 (1H-1), pp.4-17–4-18(1994).

[6] 藤井, 他: Keio-MMP における連続メディアオブジェクトへのアプローチ, 信学技報 CPSY-96-112, pp.1–8. (1997).

[7] IEEE: *Threads Extension for Portable Operating Systems (Draft 6),* P1003.4a/D6 (1992).

[8] ISO/IEC: *POSIX Part 1: System Application Program Interface (API) [C Language]*, ISO/IEC 9945-1 (ANSI/IEEE Std 1003.1) (1996)