

感性を考慮したジョブスケジューリング

栗田 亮[†] 千葉 滋[†] 光来 健一^{††}

本稿では重要性の低いマルチメディア処理に対してユーザの感性を考慮した QoS 処理を行う新しいスケジューリングシステム *Tanma* について述べる。マルチメディア処理に対する従来の QoS 処理は、重要性の低いプロセスを犠牲にして、常に一定以上の計算資源を割り当てるためのものであった。しかし近年では、BGM のように状況によっては処理を持続できなくても構わないマルチメディア処理が増えている。より重要性の高いプロセスが存在する場合は、そのような重要性の低い処理を行うプロセスから計算資源を迅速に移さなければならないが、急激に移し過ぎるとマルチメディア処理の場合、ユーザが不快に感じることがある。*Tanma* はこのような不快感を避けるように音楽再生プロセスの資源の割り当てを行う。*Tanma* は progress-based regulation と呼ばれる手法をもとにスケジューリングを行うが、音楽再生プロセスから資源を奪う場合には、再生中の音楽のフェードイン/アウトのような処理を行い、ユーザが不快感を感じないようにする。

A Pleasant Job Scheduling System

RYO KURITA,[†] SHIGERU CHIBA[†] and KENICHI KOURAI^{††}

This paper presents a new scheduling system called *Tanma*, which provides new kind of QoS for low-importance multimedia processes. Traditional QoS management ensures that high-importance multimedia processes can run with higher priority than other processes. However, such traditional QoS is not suitable for low-importance multimedia processes such as a back-ground music player. Although resources should be deallocated from the multimedia processes if higher priority processes are running, immediate deallocation from these multimedia processes may give the users unpleasant feeling. *Tanma* can schedule resources among processes to avoid such unpleasant feeling when an audio player is running. *Tanma* performs the progress-based scheduling but it also performs fade-in/out of the played music when resources are deallocated from the process of the audio player.

1. はじめに

従来マルチメディア処理はその時間的制約の特徴から優先すべき処理とされ、品質を損なわないように QoS (Quality of Service) 処理が開発されてきた¹⁾²⁾³⁾。従来の QoS 処理はマシン上の CPU やディスクなどの計算資源を確保したり、資源が競合した場合にはサンプルレートを下げることで、マルチメディア処理の連続性が失われないようにすることであった。

しかし近年では、マルチメディア処理は必ずしも優先する必要のないものとして考えられる場合があり、そのようなマルチメディア処理に対する新しい QoS

処理が必要とされている。例えばバックグラウンドで行われるマルチメディア処理が挙げられる。マシン上で音楽を聴きながら他の処理をするユーザが近年では多く見られ、中にはテレビを多画面で見るとマシン上で動画を再生しながら別の仕事をするユーザもいる。このような環境の中では、マルチメディア処理とユーザの仕事が互いに資源を奪い合い、十分な資源が得られず共に処理が遅れてしまう場合がある。バックグラウンドで行われるマルチメディア処理よりユーザの仕事が重要な場合には、ユーザの仕事に優先的に資源を割り当てたいが、資源の割り当てを急激に変更するとマルチメディア処理の従来の意味での QoS が損なわれる恐れがある。

また、別の例として、いろいろなマルチメディア処理の同時実行が挙げられる。マルチメディアアプリケーションは多様化し、いろいろな種類のマルチメディア処理を同時に行うようになった。ここで各マルチメディア処理には優先順位をつける必要があり、場合によっ

[†] 東京工業大学情報理工学研究所数理・計算科学専攻
Dept. of Mathematical and Computer Sciences, Tokyo
Institute of Technology

^{††} 日本電信電話株式会社 NTT 未来ねっと研究所
NTT Network Innovation Laboratories
現在、東京工業大学情報理工学研究所数理・計算科学専攻

ては優先順位の低いマルチメディア処理を停止させなければならない。例えば動画再生をするプレイヤーは、音楽を途切れることなく処理するために動画の処理を停止させる場合がある。このような場合、優先順位の低いマルチメディア処理に対する QoS 処理は従来から開発されているが、優先順位の低いマルチメディア処理に対する QoS 処理は現在考えられてはいない。

そこで我々はこれらのような重要性の低いマルチメディア処理に対して、ユーザの感性を考慮した QoS 処理を行うジョブスケジューリング手法を提案する。我々のスケジューリング手法は progress-based regulation をベースにし、重要性の低いマルチメディア処理がユーザに不快感を与えないようにする。progress-based regulation⁴⁾ とはプロセスの進捗状況を調べて資源の競合を判断し、資源の割り当てを動的に変更するスケジューリング手法である。我々はこの手法をマルチメディア処理に適した形に改良して用い、重要性の低いマルチメディア処理の進捗が思わしくない場合は、重要性の高いプロセスと資源の競合が生じていると判断してマルチメディア処理を停止させる。これによりマルチメディア処理に使われている資源を解放することができ、重要性の高いプロセスを優先して処理することが可能である。この際に、マルチメディアデータが途切れながら再生されることのないように即座に停止させることにより、不快感をユーザに与えないようにする。その一方で、マルチメディア処理を突然停止させることでユーザを驚かせたり不安に感じさせることのないように、処理を停止させる際にマルチメディア特有の QoS 処理を行う。

我々は提案するスケジューリング手法を実現するシステム Tanma を Linux 上に実装した。今回実装した Tanma は音楽の再生処理を制御する。Tanma は重要性の低い音楽再生プロセスが使用するカーネル内のサウンドバッファを監視し、バッファに保持されているデータ量から進捗が思わしくないと判断した場合、音楽再生を一定時間停止させる。停止させる際には音楽をフェードアウトさせ、再開させる際には音楽をフェードインさせることにより、突然音楽が途切れたり再開されることを防ぐ。さらに、フェードアウト前の部分から音楽が再開されるようにすることで(タイムマシン再生)、フェードアウトが原因でユーザが音楽の一部を聴くことができないという事態を防ぐ。Tanma はカーネル内に実装されておりアプリケーションを書き換える必要がないので、既存の音楽プレイヤー全てに適用することが可能である。このシステムについて性能を調べた結果、音楽再生プロセスがユーザの感性

を考慮して制御されても、重要性の高いプロセスはほとんど遅れることなく処理された。

以下では、まず 2 章で従来の手法とその問題点について述べ、3 章で我々が提案する QoS 処理を実現するスケジューリングシステム Tanma について詳しく述べる。4 章では Tanma に関連する実験を行い、5 章で関連する研究について紹介し、6 章で本稿をまとめる。

2. 重要性の低いマルチメディア処理に対する QoS 処理

2.1 QoS の要件

重要性の低いマルチメディア処理とは、リアルタイムに行われる処理が持続できなくても構わないような処理である。例えば、BGM アプリケーションは重要なアプリケーションが動いている間は動かなくても問題はない。このような重要性の低いマルチメディア処理に対する QoS 処理とは、ユーザの感性を考慮すること、すなわち、ユーザに不快感を与えないようにすることである。

まず考えられる QoS 処理は、重要性の低いマルチメディア処理がユーザに不快感を与えそうになった場合、瞬時に止めることである。これによりマルチメディア処理がユーザに与える不快感を最小限に抑えることができる。上で挙げた BGM の例の場合、BGM 処理が資源不足に陥り音が途切れてしまう問題がある。一瞬であれば問題はないが、重要性の高いプロセスが競合する資源を長時間、大量に使用する場合は途切れる度合いが大きくなってしまふ。頻繁な途切れが生じる音楽はユーザにとっては聞き苦しいものなので、このような場合は音楽を止める方がよい。そうすることで資源の解放にもなり、重要性の高いユーザの仕事を速く終わらせることができる。

次に、マルチメディア処理を停止および再開させる際に、ユーザに不快感を与えないようにする必要がある。突然停止や再開をさせると、ユーザを驚かせてしまったり、システムに不具合があるのではないかと不安にさせてしまふ。マルチメディア処理をユーザの指示によらずに停止および再開させる際には、その前にそのことをユーザに知らせる QoS 処理が必要である。この QoS 処理にはメディアごとに様々な形が存在する。音楽を停止させる場合は、その前後で音楽をフェードアウト/インさせることにより、ユーザに違和感なく停止および再開させることができる。動画の場合は、動画を暗転させたりモザイクをかけていくなどのフェードアウト/インを行ったり、停止中に「しばらくお待ちください」と表示させるなど、音楽より

も様々な QoS 処理が可能である。複数のウィンドウを表示するアプリケーションでは、使用していないウィンドウを徐々に小さくして画像処理を減らすことが考えられる。

2.2 従来の手法の問題点

前節で述べた新しい QoS 処理は従来の手法では実現することは難しい。この QoS 処理を提供するには (1) ユーザに不快感を与えそうになった場合に停止させること (2) 突然停止させることによりユーザに不快感を与えないことの 2 つの処理が実行可能でなければならない。CPU スケジューリングや、プロセスの進捗状況をもとに資源の割り当てを変更する従来の手法が抱える問題点についてそれぞれ説明する。

2.2.1 CPU スケジューリング

従来のスケジューリングは CPU を割り当てる時間を制御するものである。Linux には CPU 優先度を変更する nice コマンドがあり、重要性の高いプロセスには CPU を多く割り当て、重要性の低いプロセスには CPU を少しだけ割り当てることができる。マルチメディア処理よりも重要性の高いプロセスに CPU を多く割り当てることで、重要性の高いプロセスを優先させて処理することが可能である。しかし、マルチメディア処理に割り当てられる CPU が一方的に減らされてしまうので、音の途切れなどの不具合が生じてしまう。

そこで、CPU の使用率を監視する方法が挙げられる⁵⁾。マルチメディア処理以外に CPU を多く使う処理が起動している場合は資源の競合が生じていると考え、マルチメディア処理を停止させ、途切れなどのマルチメディア処理の不具合を起こさせないようにする。しかし、この方法で処理を停止させてよいか判断するのは難しい。なぜなら競合が起こる資源は CPU だけではないからである。CPU の使用時間が短くても他の資源を多く使用する処理に対しては、CPU の使用率を見るだけでは停止させてよいかどうかを判断することはできない。CPU の使用時間が長くても他の資源をほとんど使用しない場合もある。このような問題に対処するには、CPU 以外にもディスクやネットワークバンド幅など全ての資源の使用状況を同時に調べなければならないが⁶⁾、実現させることは困難である⁴⁾。

2.2.2 Progress-based regulation

近年 progress-based regulation という新しいスケジューリング手法が注目されている。これはプロセスの進捗状況に応じて資源の割り当てを変更する手法である。重要性の低いプロセスの処理の進捗を調べ、遅

れていれば資源の競合が生じていると判断し、重要性の低いプロセスを一時停止させる。

だが progress-based regulation では重要性の低いプロセスに対する QoS 処理は全く考えられていない。progress-based regulation は重要性の低いプロセスに注目はしているが、あくまで重要性の高いプロセスのための QoS 処理を行うことが目的である。そのため重要性の低いプロセスがマルチメディア処理の場合、マルチメディア処理の QoS が損なわれてユーザに不快感を与えてしまう。progress-based regulation に基づくシステム MS Manners⁴⁾ が対象としているバックアップやディスクデフラグはユーザから見えない処理であったので問題はなかったが、マルチメディア処理はユーザから見える処理であることが問題を生みだしている。

まず挙げられる問題は、progress-based regulation がマルチメディア処理を突然停止および再開させることである。重要性の高いプロセスと資源の競合が生じた場合、重要性の低いプロセスはすぐに停止させられ、競合が解消されればすぐに再開されてしまう。この手法を重要性の低いマルチメディア処理に対して利用すれば、マルチメディア処理を突然停止および再開させることになり、ユーザを驚かせて不安にさせてしまう。

次に問題なことは、progress-based regulation が行う進捗状況を調べるための一時的な処理再開である。progress-based regulation は一定時間停止した後に進捗状況を判断するまで少しの間だけ処理を再開させ進捗状況を調べるが、マルチメディア処理ではこの一時的な再開が問題である。わずかな時間だけマルチメディア処理を実行すると、ユーザにとっては短すぎて価値のない処理なのでユーザに不快感を与えてしまい、さらにその短い再生の分だけ再開する場所が先に進んでしまう。

また、従来の progress-based regulation には停止時間の設定方法にも問題がある。MS Manners では重要性の低いプロセスを停止させた後、進捗が思わしくない場合はまだしばらくは重要性の高いプロセスは終了しないと判断し、停止時間を指数関数的に増やしていく。それゆえ重要性の高いプロセスが終了してもしばらく停止したままとなるが、ユーザから見えない処理なので問題視されなかった。しかしマルチメディア処理の場合、重要な仕事が終了した後すぐに再開されないとユーザを苛立たせ不安にさせてしまう。

3. Tanma

我々は重要性の低い音楽再生プロセスに対し QoS

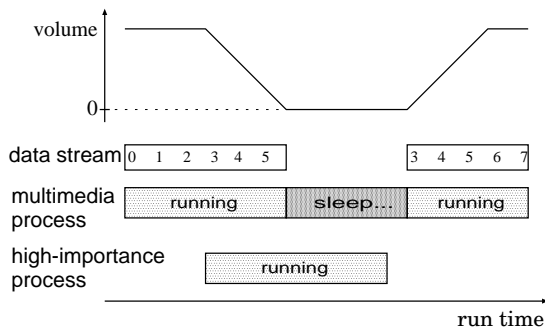


図 1 Tanma の処理の流れ

処理を行うスケジューリングシステム Tanma を開発した。Tanma は資源が競合した場合に BGM のような重要性の低いプロセスによるユーザへの不快感を抑えるために、ユーザの感性を考慮した新しい QoS 処理を音楽プレーヤーに施すべく制御する。Tanma は progress-based regulation により重要性の低い音楽プレーヤーを進捗状況に応じて停止させることで、音の途切れを未然に防ぐことが可能である。また、この停止および再開によるユーザへの不快感を抑えるために、音量の自動調整とタイムマシン再生を行う。これらの機能を使うことでユーザに不快感を与えることなく音楽を停止および再開させることができる。

Tanma による音楽再生プロセスの制御は図 1 のような流れで行われる。Tanma は音楽再生プロセスの進捗を調べて遅れが生じているかどうかを判断する。遅れが生じている場合は重要性の高いプロセスと資源の競合を起こしていると判断し、音量の自動調整機能により音量を徐々に下げ、音量が 0 になれば音楽を一定時間停止させる。一定時間停止させた後、まだ遅れが生じる場合は停止を続け、問題ない場合は音量の自動調整機能により音量を徐々に上げながら音楽を再開させる。この際、タイムマシン再生機能によりフェードアウトを始めた場所から音楽を再開させる。この機能を使うことで、音量の自動調整機能によって聞こえにくくなってしまった部分を改めて再生しなおすことが可能である。

Tanma のスケジューリングは MS Manners のようにアプリケーションから進捗情報を得るのではなく、カーネルからアプリケーションを監視することで行われる。これにより、全ての音楽アプリケーションに対処することが可能である。

3.1 スケジューリング

progress-based regulation にもとづき、音楽再生プロセスの進捗状況に応じて資源の割り当てを変更する。音が途切れそうになればそのプロセスを止めるこ

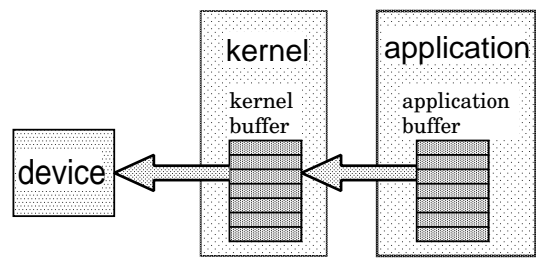


図 2 サウンドデータの転送

とにより、不具合をできるだけユーザにみせることなく、重要性の高いプロセスに資源を割り当てることができる。

進捗状況の判断には再生データを格納するカーネルバッファを調べることで行う。音楽の再生は図 2 のようにアプリケーションからカーネル、カーネルからサウンドデバイスとサウンドデータが転送されることによって行われる⁷⁾。音楽再生プロセスによってアプリケーションからこのカーネルバッファにデータが転送されることで、カーネルバッファ内のデータは増えていく。これに対しカーネルバッファ内のデータはデバイスに DMA 転送され、バッファ内のデータは徐々に減っていく。音楽再生プロセスに資源があまり割り当てられなくなると、アプリケーションからカーネルへのデータ転送が遅れてしまう。これによりカーネルバッファ内のデータが減っていき、場合によっては DMA 転送するデータがなくなり音が途切れてしまう。

Tanma のスケジューラはカーネルバッファ内に保持されている再生データのサイズを定期的に調べ、データサイズが閾値を下回ると進捗が遅れていると判断する。音楽再生プロセスと同時に資源を多く消費する重要性の高いプロセスが起動すると、音楽再生プロセスが遅れてしまいバッファ内のデータサイズが閾値よりも小さくなるからである。この場合は資源の競合が生じていると判断し、音楽再生プロセスを停止させる。一定時間たてば再び進捗状況を調べ、音楽再生プロセスの資源がまだ競合していれば、閾値よりバッファ内のデータサイズが小さいままとなるので停止を続ける。音楽再生プロセスの資源の競合が解消されればバッファ内のデータサイズは閾値より大きくなるので、この場合は音楽を再開させる。

停止時間はユーザの感性と処理性能のバランスを考え最大 16 秒でランダムに設定する。MS Manners では停止時間を指数関数的に増加させていたが、Tanma では停止時間に上限を設けることで、重要性の高いプロセスの終了後、音楽プロセスが長くとも十数秒停止した後に再開されるようにした。この程度の停止時間

であればユーザは待たされても不満に思うことはないと考えた。

3.2 停止・再開に伴う機能

Tanma は音楽の停止および再開によりユーザを驚かせ不安にさせることのないように、QoS 処理を提供する 2 つの機能を備えている。これらの機能はスケジューリングと連動しており、音楽の停止および再開とあわせて動的に実行される。

音量の自動調整

音量の自動調整機能は、音楽を停止させる前後で音量を調整することで、ユーザに与える不快感を取り除く機能である。音楽を停止させる必要がある場合にこの音量の自動調整機能が働くことにより、ユーザに停止および再開させることを自然に知らせることができる。停止させる前にフェードアウトさせながら音楽を再生させ、停止させた後に再開する時はフェードインさせながら音楽を再生させる。フェードアウト/インは一定の周期で音量を徐々に変化させることで行われる。フェードアウトは音量が 0 になるまで、フェードインは音量が元の値に戻るまで行われる。フェードアウトは進捗が思わしくない場合に行われ、フェードインは進捗が問題ない場合に行われるが、進捗状況の変化に応じてフェードアウトからフェードイン、フェードインからフェードアウトに切り替えることが可能である。

また、音量の自動調整機能により progress-based regulation の問題点であった一定時間停止させた後の短い再生に対処することができる。progress-based regulation では一定時間停止し終わった後に進捗状況を調べるために短い時間だけ音楽再生をしなければならず、これがユーザに不快感を与えてしまう。音量の自動調整機能は、この短い時間の再生中の音量を 0 にしてユーザには聞こえないようにする。これによりユーザへの不快感を抑えることが可能である。

タイムマシン再生

タイムマシン再生機能は音楽の再開時に、フェードアウト前の部分から音楽を再生させる機能である。フェードアウトを行うことにより音量が小さくユーザには聞こえにくい部分が出てしまう。この部分をユーザが聞くことができるように、音楽が再開される際にその部分から改めて再生を行う。進捗が思わしくない場合に音楽のフェードアウトを行うが、このフェードアウト中に再生しているデータをカーネル内に保存する。このデータ保存は音量が 0 になって停止状態に入るか、途中で進捗が良好となり通常の再生状態に戻るまで続けられる。音量が 0 になり停止した後、進捗が問題な

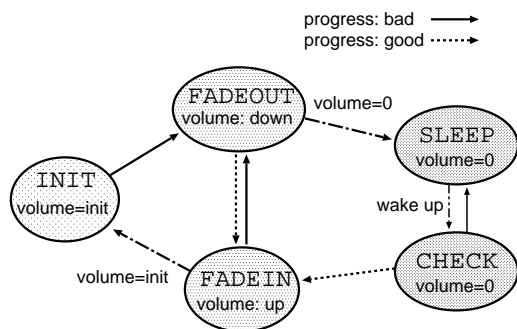


図 3 Tanma における音楽再生プロセスの状態遷移

いと判断されればフェードインしながら保存していたデータを再生する。保存していたデータが全て再生された後で、フェードアウト後の位置から続けて音楽を再生する。

3.3 実装

以上述べてきた音楽再生に QoS 処理を施し制御するスケジューリングシステム Tanma を Linux 2.4.20 に実装した。サウンドカード Ensoniq AudioPCI (ES1371) のデバイスドライバ es1371.c に実装を施し、音楽の再生処理を行う関数 es1371_write() 内でスケジューリング、音量の自動調整、タイムマシン再生を実行する。これにより全ての音楽再生プロセスの進捗状況が Tanma によって監視される。音楽再生プロセスは基本的には重要性の低いプロセスとして扱われ、他のプロセスと資源が競合して処理に遅延が生じれば Tanma により制御される。音楽再生プロセスの重要性を高く設定し、Tanma によって制御されないようにするには、nice コマンドを使い音楽再生プロセスの優先度を上げればよい。

Tanma による音楽再生プロセスの制御は図 3 で示す状態遷移によって行われる。通常、プロセスは進捗状況に問題がなければ INIT 状態にある。進捗が思わしくない場合は INIT 状態から FADEOUT 状態に移る。FADEOUT 状態では徐々に音量を下げていき、音量が 0 になった時点で SLEEP 状態に移り、一定時間停止する。一定時間停止した後は CHECK 状態に移り、進捗が思わしくなければ SLEEP 状態に移り停止を続け、進捗が問題なければ FADEIN 状態に移る。FADEIN 状態では徐々に音量を上げていき、音量が元の値に戻った時点で INIT 状態に移る。FADEOUT, FADEIN 状態でも進捗状況を監視しているので FADEOUT 状態から FADEIN 状態、FADEIN 状態から FADEOUT 状態のようにプロセスの状態が遷移する場合もある。

スケジューリング

進捗状況はカーネルバッファ内に格納されている再生データのサイズを調べることで判断する。一定の周期でデータサイズと閾値を比較し、データサイズが閾値より小さい場合は進捗が思わしくないと判断する。この周期は MS Manners と同様に 0.5 秒とした。閾値は過去調べたデータサイズの平均値をもとに設定される。停止には `schedule_timeout()` を呼び、8 秒から 16 秒のランダムな時間だけ停止する。停止が終わった後も 0.5 秒かけて進捗状況を調べ、進捗がまだ思わしくない場合は停止を続ける。

音量の自動調整

音量の設定は同じデバイスドライバにある関数 `mixdev_ioctl()` を呼び、音量を変更する周期はスケジューリングと同様に 0.5 秒ごとに行う。フェードアウト/インを実行する時間は、音量の変化による違和感をユーザになるべく感じさせないように 3 秒に設定した。対象とする音量はマスタ音量である。INIT 状態から FADEOUT 状態に移る際に、その時点での音量の値を初期値として取得しておく。この値は FADEIN 状態から INIT 状態に戻る際に用いられる。

タイムマシン再生

INIT 状態から FADEOUT 状態までの間に再生されるデータを保存するため、カーネル内にリングバッファを用意した。このバッファのサイズは、ユーザの感性を考え 5 秒分の再生データを保存できる値に設定した。3 秒かけてフェードアウトし平均して 12 秒間の停止と 0.5 秒間の進捗状況確認を繰り返すと仮定すると、継続した停止時間が 50 秒以内ならばこのバッファが溢れることはない。それ以上停止を継続した場合は古いデータから少しずつ破棄されてしまうが、音楽の重要性が低いので長時間停止した場合は、どこまで再生されていたかをユーザは意識しなくなると考えられる。そのため、始めの部分の再生ができなくてもそれほど問題ではないであろう。

4. 実験

4.1 動作検証

今回実装した Tanma は音楽再生プロセスを対象にしている。実装はカーネルにのみ行っているのでアプリケーションを書き換える必要はない。これにより全ての音楽アプリケーションを制御することが可能である。実際に検証したところ、wav プレーヤーや mp3 プレーヤーでも問題なく動作した。Tanma はマスタ音量を変更しているため、CD や MIDI の音楽制御も可能である。動画ファイルである mpeg 再生に関して

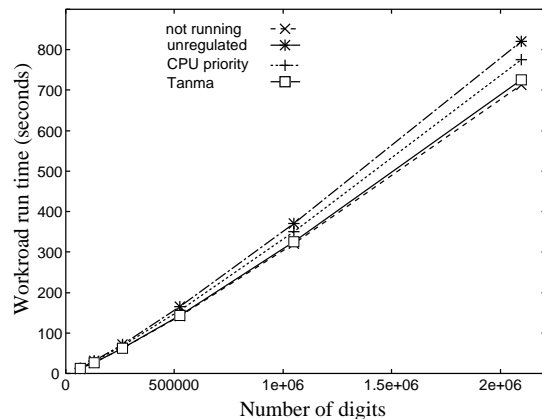


図 4 計算の処理時間

も、動画は突然停止するが音楽に対しては Tanma が提供するフェードアウト/インなどの QoS 処理を行うことができた。

4.2 性能評価

我々が開発した Tanma に関して性能評価とオーバーヘッド測定を行った。音楽再生プロセスと重要性の高いプロセスを同時に実行した場合、Tanma がどのような影響を及ぼすかを調べた。実験には CPU が Pentium 733MHz、メモリが 512MB の計算機を使用し、Linux 2.4.20 をベースにした Tanma 上で測定した。音楽再生プロセスには音楽プレーヤーである `realplayer`⁸⁾ を使用し、これと同時に実行する重要性の高い処理には、常に一定の負荷がかかる 計算アプリケーション `pi_fftc`⁹⁾ を使用した。

4.2.1 資源の競合に対する Tanma の性能検証

今回実装した Tanma は音楽再生プロセスによる資源の競合を防ぐことが可能である。そこで、音楽再生プロセスの実行中に重要性の高いプロセスを実行するという状況で、重要性の高いプロセスの性能を調べるための実験を行った。音楽プレーヤーとの資源の競合による重要性の高いプロセスの遅延を、Tanma を用いることでどの程度減らすことができるかを調べた。

図 4 は音楽プレーヤーが動作している状態での計算にかかった時間を示す。この値を 200 万桁まで計算する時間は、他にプロセスが実行されていない場合に 713 秒かかった。通常システム下で、音楽プレーヤーを動作させながら実行した 計算にかかった時間は、資源の競合が生じ 821 秒となり、15%多く時間がかかった。CPU 優先度を変更し 計算を優先的に実行したところ 776 秒かかり、9%の遅延が残った。これは CPU の競合が完全にはなくなっていないが、あるいは CPU 以外の資源の競合が生じていることが原因と

考えられる。このとき音楽プレーヤーには十分な資源が割り当てられず、音の途切れが激しいものとなった。これらに対し Tanma を有効にして CPU 優先度を変更した。計算を実行したところ 725 秒となり、1.7% のオーバーヘッドであった。音楽プレーヤーは計算の実行開始時に Tanma により停止させられ、計算が終了するまで 730 秒間停止を継続した。この間に進捗状況の判定のための一時的な処理再開は平均 56 回行われたが、ユーザ側からは停止し続けているようにみえた。この音楽プレーヤーの制御により、計算だけを実行する場合とほぼ変わらない時間で処理を終えることができた。

また、Tanma と MS Manners とで性能の比較を行った。Tanma では音楽のフェードアウトを行う時間だけ音楽プレーヤーを止める時間が遅れてしまう。また、停止時間が MS Manners のように初期値 8 秒から進捗状況を判定するたびに倍に設定されていくのではなく、8 秒から 16 秒の乱数に設定されるので、Tanma ではより多く進捗状況の判定が行われる。これらの違いによる計算の処理時間の差があるかどうかを調べた。MS Manners の方式が動作している状態での計算にかかる時間を測定した結果、この値を 200 万桁まで計算する時間は 722 秒となり、1.3% のオーバーヘッドであった。これは Tanma 上での計算時間とほぼ同じ値となったので、MS Manners の方式との違いによる遅れはほとんどないことがわかった。ちなみに MS Manners 方式でも音楽プレーヤーは計算の実行開始から終了まで停止を継続した。この間の進捗状況の判定は平均 6 回行われた。

4.2.2 Tanma のオーバーヘッド

Tanma では進捗状況の監視と判定、音量の自動調整、タイムマシン再生を行うが、これらの処理によるオーバーヘッドを測定した。音楽再生と計算を同時に行い、音楽再生の進捗状況の指標であるカーネルバッファ内の再生データサイズの測定を行う。Tanma の処理オーバーヘッドによりアプリケーションプロセスに割り当て可能な資源が減ってしまった場合は、アプリケーションからカーネルへの再生データの転送に遅延が生じ、カーネルバッファ内の再生データが減ってしまうはずである。データサイズの測定は 700 秒間行い、バッファ内のデータの増減がオーバーヘッドにより変化するのかどうかを調べた。カーネルバッファのサイズは 131,072 バイトであった。実験は (1) 通常のシステム (2) 進捗状況の監視と判定のみ行う Tanma (3) 音楽のフェードアウト/インを繰り返す Tanma、これら 3 つの環境下で実行し比較した (2) ではカーネル

表 1 カーネルバッファ内のデータサイズの測定結果

	平均値 (byte)	標準偏差 (byte)	最小値 (byte)
(1) 通常のシステム	115846	6839	98336
(2) 進捗状況の監視と判定のみ行う Tanma	115246	7201	98336
(3) フェードアウト/インを繰り返す Tanma	111947	4713	98336

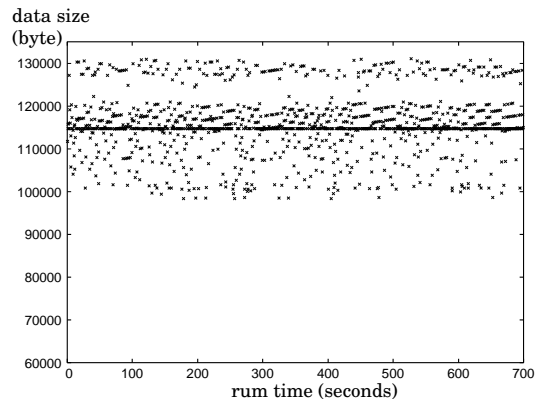


図 5 (1) 通常のシステム上でのカーネルバッファ内のデータサイズ

バッファ内のデータサイズを取得し閾値と比較する処理を行う。ただしバッファ内のデータサイズが閾値より小さい場合でも、音量の自動調整やタイムマシン再生で行われる処理は実行しないものとした (3) では本来は進捗状況が変化したときに行われる音量の自動調整やタイムマシン再生によるオーバーヘッドを測定するために、これらの処理を繰り返し実行する。

実験を行った結果、表 1 で示す値が測定された (1) と (2) をそれぞれ比較してみると、どの値も大差ないことから進捗状況の監視と判定によるオーバーヘッドはほとんどないことがいえる。図 5 と図 6 は (1) と (2) のデータサイズの分布をそれぞれ表すが、この 2 つの図を見比べても大差ないことがわかる。それに対し (3) の環境下での値を比較してみると、平均値が (1) の場合よりも小さいことからオーバーヘッドにより遅延が生じている。図 7 は (3) の環境下でのデータサイズの分布を示すが、この図をみても (1) と比べて小さい値に多く分布していることがわかる。しかしデータサイズの最小値は (1)(2)(3) 共に同じ値となったので、Tanma のオーバーヘッドが原因でデータ転送が長い時間遅れてしまうことはない。このことから Tanma によるオーバーヘッドは問題視するほど大きいものではない。

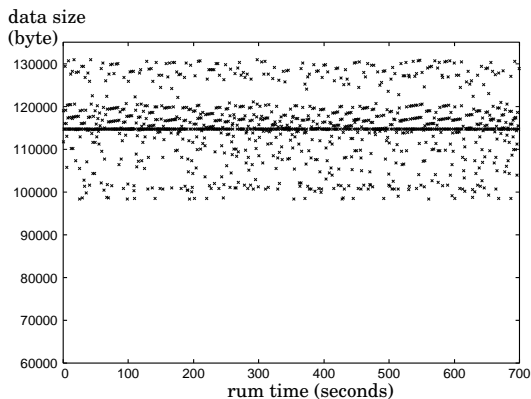


図 6 (2) 進捗状況の監視と判定のみ行う Tanma 上でのカーネルバッファ内のデータサイズ

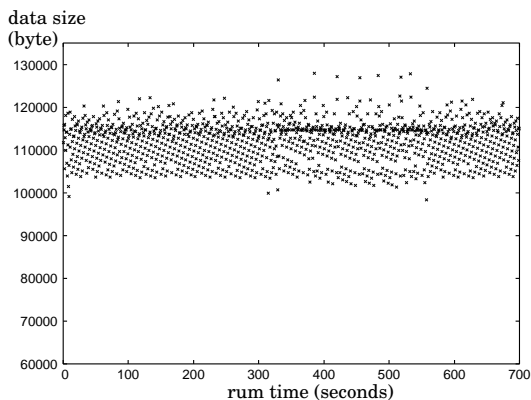


図 7 (3) フェードアウト/インを繰り返す Tanma 上でのカーネルバッファ内のデータサイズ

5. 関連研究

QoS adaptation/real-rate scheduling¹⁰⁾ はマルチメディアアプリケーションに対し、進捗状況を利用した動的な資源の割り当てと QoS 処理を行うシステムである。メディアデータの各パケットにタイムスタンプとプライオリティを付加する。バッファ内のパケットに付いているタイムスタンプをもとに進捗状況を調べ、進捗状況に応じて資源の割り当てを変更しマルチメディア処理に十分な資源を与える。急激な進捗状況の変化により QoS が保証できなくなりそうな場合は、バッファ内にあるプライオリティの低いパケットを破棄することにより QoS を保証する。Tanma と同様にマルチメディア処理の進捗状況に応じて動的に資源の割り当てを変更するシステムである。しかし、この研究は従来のマルチメディア向けのリアルタイム OS と同様に、重要性の高いマルチメディア処理に対する QoS 処理を目的としているので、今回の目的である重

要性の低いマルチメディア処理に対する QoS 処理とは方向性が異なる。

6. まとめ

本稿では、重要性の低いマルチメディア処理に対し QoS 処理を行うジョブスケジューリングシステム Tanma について述べた。重要性の低いマルチメディア処理の資源を急激に奪うことでユーザに不快感を与えないよう、ユーザの感性を考慮した新しい QoS 処理を行う。Tanma では progress-based regulation をマルチメディア処理に適した形にして用い、再生中の音楽のフェードアウト/インなどの処理を行うことで、ユーザに不快感を与えることなく音楽再生プロセスをすばやく制御することができる。我々が行った実験の結果、Tanma を使うことにより音楽再生プロセスと重要性の高いプロセスとの資源の競合を制御することができた。

今回実装した Tanma は停止状態になるとユーザ側からの指示を停止が終わるまで受け付けなくなる場合がある。例えば停止中に一時停止ボタンを押したり閉じるボタンを押しても音楽プレイヤーは反応せず、停止状態が解除されてからはじめてユーザ側の指示を実行する。これは音楽プレイヤーが単一のプロセスで動作しているのが原因である。音楽再生や停止など全ての命令が同じプロセス上で行われているので、音楽再生のプロセスを停止させることで一部の動作が停止されてしまう。この問題に対処できるようにすることが今後の課題である。

また、今回実装した Tanma では複数の音楽プレイヤーの中から特定のものだけを制御することはできない。重要性の異なる 2 つの音楽プレイヤーが同時に起動している場合、Tanma が監視するサウンドバッファに 2 つの音楽プレイヤーが共にサウンドデータを転送するので、それぞれの進捗状況を調べることはできず、どの音楽プレイヤーを制御しなければならないのか判断することができない。この問題に対処するには、サウンドデバイスを複数作成できるようにして、各音楽プレイヤーが専用のサウンドバッファにサウンドデータを転送する方法が考えられる。また、音楽の混在はユーザに不快感を与える場合が多いので、重要性の低い音楽プレイヤーを停止させた後に重要性の高い音楽プレイヤーを起動させる方法も考えられる。

参考文献

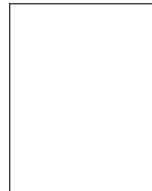
- 1) Childs, S. and Ingram, D.: The Linux-SRT integrated multimedia operating system: Bring-

- ing QoS to the desktop, *In Proceedings of 7th Real-Time Technology and Applications Symposium* (2001).
- 2) Rajkumar, R., Juvva, K., Molano, A. and Oikawa, S.: Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems, *In Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking*, pp. 150–164 (1998).
 - 3) Sundaram, V., Chandra, A., Goyal, P., Shenoy, P. J., Sahni, J. and Vin, H. M.: Application Performance in the QLinux Multimedia Operating System, *In Proceedings of 8th ACM Conference on Multimedia* (2000).
 - 4) Douceur, J. R. and Bolosky, W. J.: Progress-based regulation of low-importance processes, *In Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pp. 247–260 (1999).
 - 5) Lampson, B. W.: Hints for computer system design, *IEEE Software* 1 (1), pp. 11–28 (1984).
 - 6) Verghese, B., Gupta, A. and Rosenblum, M.: Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors, *In Proceedings of the 8th ASPLOS*, pp. 181–192 (1998).
 - 7) Jeff Tranter 著, 山形 浩生 訳: Linux マルチメディアガイド, O'Reilly Japan (1997).
 - 8) Real.com: RealNetworks, <http://www.jp.real.com>.
 - 9) Ooura, T.: FFT と AGM による円周率計算プログラム, http://www.kurims.kyoto-u.ac.jp/~ooura/pi_fft-j.html.
 - 10) Goel, A., Shor, M. H., Walpole, J., Steere, D. and Pu, C.: Using Feedback Control for a Network and CPU Resource Management Application, *In Proceedings of 2001 American Control Conference* (2001).

(平成 ? 年 ? 月 ? 日受付)

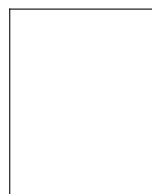
(平成 ? 年 ? 月 ? 日採録)

(担当編集委員 中島 浩)



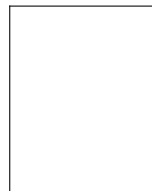
栗田 亮

1980 年生。2002 年東京工業大学理学部情報科学科卒業。現在同大学院情報理工学研究科数理・計算科学専攻修士課程。オペレーティングシステム, マルチメディア処理の研究に従事。日本ソフトウェア科学会会員。



千葉 滋 (正会員)

1968 年生。1991 年東京大学理学部情報科学科卒業。1993 年同大学院理学系研究科情報科学専攻修士課程修了。1996 年同専攻博士 (理学) 取得。同専攻助手, 筑波大学電子・情報工学系講師, 東京工業大学情報理工学研究科講師を経て, 現在同助教授。言語処理系およびオペレーティングシステムなどシステムソフトウェアの研究に従事。情報処理学会, 日本ソフトウェア科学会, ACM 各会員。



光来 健一 (正会員)

1975 年生。2002 年東京大学大学院理学系研究科情報科学専攻博士課程修了。日本電信電話株式会社勤務を経て, 2003 年から東京工業大学情報理工学研究科数理・計算科学専攻助手。博士 (理学)。オペレーティングシステム, ネットワークの研究に従事。日本ソフトウェア科学会, ACM 各会員。