2008－OS－109 (14)
2008／8／7

システムコール発行間隔の制御による プロセッサ使用量の調整

境 講一[†] 田端 利宏[†] 谷口 秀夫[†] 箱守 聰[‡]
[†]岡山大学大学院自然科学研究科 [‡]株式会社 NTT データ技術開発本部

計算機ハードウェアの性能向上は，ソフトウェアの処理時間を短縮させ，複雑な処理を可能にしている．一方，ソフトウェアの実行性能はハードウェア性能に大きく依存する．このため，ハードウェア性能の範囲でプログラム実行速度を調整制御する制御方式の確立が必要である．本稿では，プロセッサ使用量を調整し，プログラム実行速度を調整するライブラリの制御法を述べる．具体的には，システムコール発行間隔を制御し，プロセッサ使用量を調整するライブラリの基本方式を述べ，プロセスの停止方法と割り当てるプロセッサ性能の指定方法について述べる．さらに，実装と評価により，本制御法の特徴と有効性を明らかにする．

Control of Processor Usage by Regulating Intervals between System Calls

Koichi Sakai[†], Toshihiro Tabata[†], Hideo Taniguchi[†] and Satoshi Hakomori[‡]
[†]Graduate School of Natural Science and Technology, Okayama University
[‡]Research and Development Headquarters, NTT Data

Improvement of hardware performance reduces the processing time of software and various processing can be executed on computers. On the other hand, processing performance of software depends on hardware performance significantly. Therefore, it is necessary to establish mechanism that regulating a program execution speed within hardware performance. This paper proposes a mechanism of library which regulates program execution speed by controlling a time of processor usage. Specifically, this paper describes the basic method of a library which controls a time of processor usage based on intervals between system-calls. This paper describes a method of stopping process and a method of assigning processor performance. Furthermore, we implement and evaluate the proposed mechanism to clarify the characteristic and the effectiveness of it.

1. はじめに

近年，計算機ハードウェアの性能は著しく向上し，処理時間の短縮や複雑な処理の実行が可能になっている．一方，ソフトウェアの実行性能は，ハードウェア性能に大きく依存する．このため，例えば，高性能な計算機と低性能な計算機では，同じソフトウェアでも表示速度が大きく異なるため，利便性が低下する．また，計算機を利用したソフトウェア教材は，アニメーションを利用し，学習者の理解を支援する環境を提供している．このようなソフトウェア教材を利用する学習者は，アニメーションでの説明を何度も，かつ希望の速度で見たいという要望がある．つまり，利用者の要求に合わせた速度でプログラム実行速度を自由に調整したいという要望がある．一方，この要望を満足する機能を各ソフトウェアに組みこむことは非常に困難である．

そこで，著者らは，計算機ハードウェア性能の範囲で，利用者が求める速度でプログラム実行速度を自由に調整する方法について，研究を進めている．現在までに，プロセスの入出力性能を調整しプログラム実行速度を調整する方法[1]，[2]，[3]，およびプロセッサ性能を調整する方法としてプロセスのスケジュール法を工夫しプログラム実行速度を調整する方法[4]，[5]，[6]を提案した．プロセスの入出力性能を調整する方法は，プロセスが OS カーネルへ要求する入出力ごとの性能を調整する．この制御法は大きく以下の 2 つに分類できる．1 つは，1 つの入出

力に要する時間を調整する方法である．もう 1 つは，単位時間における入出力の回数を調整する方法である．これらの制御法を，OS カーネル内へ実装する方式(以降，OS 方式と呼ぶ)と OS カーネル外にライブラリとして実装する方式(以降，ライブラリ方式と呼ぶ)の両方式で実装し，比較評価を行った．

一方，プロセッサ性能を調整する方法は，プロセスに割り当てるプロセッサの割合を調整することでプログラム実行速度を調整する．この制御法は，プロセッサの実行を単位時間(以降，タイムスロットと呼ぶ)に分割し，プロセス処理にタイムスロットを割り当てる割合を調整する．この制御法は，プロセスのスケジュール法を工夫するため，OS 方式での実装が好ましく，ライブラリ方式で実装することは難しい．

制御方式を OS 方式で実装する場合，応用プログラム(以降，AP と略す)の実行ファイルをそのまま使用できるという利点を持つ．一方，ライブラリ方式で実装できれば，OS カーネルの作成環境が不要になるため，多くの OS 上でプログラム実行速度を調整できるようになる．

本稿では，システムコールの発行間隔を制御し，プログラム実行速度を調整する方法を提案し，ライブラリとして実装する．具体的には，プロセスのシステムコール発行間隔に着目し，プロセスがシステムコールを発行した際，その実行直前に強制的に走行を停止させることでシステムコールの発行間隔を制御する．これにより，プログラムの実行速度を調整する．この方法をライブラリとして実装した．また，試作したライブラリについて，オーバヘッド，調整の限界，および調整の精度の観点で評価した．さらに，共存プロセスが複数存在する場合も評価した．

従来，プロセススケジューリングについての研究は，I/O 要求の順番を入れ替えることでスループットを調整し，ディスク性能を向上させる研究[7]や，実時間性を保証する I/O スケジューリング法の研究[8]，[9]，および CPU 使用の予約と使用時間の制御によりスケジューリングを予想可能にする研究[10]が行われている．これらの研究は，ハードウェア性能を最大限に引き出す．これに対し，本研究は，要求された処理性能分だけの性能をプロセスに提供し，実行速度を調整する．

2. プログラム実行速度の調整

2.1 ライブラリ実装への要求と要望

プログラム実行速度の調整をライブラリ方式で実装する際，以下の要求と要望がある．

(要求1) 実装箇所の局所化：ライブラリ方式として実装するため，制御機構の実装はライブラリ内に閉じることが要求される．これにより，既存の OS や AP の変更が不要であり，多くの環境でプログラム実行速度の調整が可能になる．

(要求2) 制御オーバヘッドの抑制：計算機には，実行速度を調整するプロセスと実行速度を調整しないプロセスが共存する．このため，実行速度を調整制御する機構のオーバヘッドを抑え，各プロセスへの影響を抑制することが要求される．

(要求3) 自由な調整制御：利用者は，実行速度を調整したいプロセス(以降，被調整プロセスと呼ぶ)を希望する速度で実行したい．また，この要求は，プロセス実行途中で実行速度を調整する要求も含んでいる．このため，プロセス実行途中に他のプロセス(以降，性能指定プロセスと呼ぶ)から自由に速度調整を行えることが要求される．

(要望1) 調整精度の向上：プロセスの実行速度を利用者の希望する速度に近づけるため，調整の精度が高いことが望まれる．

2.2 基本方式

基本的な方式として，プロセスがプロセッサを使用した量を計測し，その使用量を調整することでプログラム実行速度を調整する．(要求 1)を満足するため，ライブラリ内で取得可能なシステムコールの発行と終了に着目する．具体的には，プロセスのシステムコール発行時刻を獲得し，システムコール処理終了から次のシステムコール発行まで連続でプロセッサを使用したと仮定し，この時間をプロセッサ使用量とする．このプロセッサ使用量を基に，利用者の指定する要求プロセッサ性能(以降，調整の程度と呼ぶ)から制御量を算出する．算出した制御量を基に，ライブラリがシステムコールを発行する直前にプロセスを停止させる．

調整の様子を図 1 に示し，以下に説明する．本方式は，調整の程度を n%とするとき，調整した場合の処理時間(T)が非調整の場合の処理時間(t)に比べ 100/n 倍になるようプロセスを停止させる．ただし，実プロセッサ性能を最高性能とするため，調整の程度の最大値は 100%である．図 1 は，調整の程度が 50%の場合である．図 1 のプロセッサ使用量 t1 に着目すると，調整時の処理時間 T1 は，

$$T1=(100/50)*t1=2*t1 \qquad (1)$$

となる．また，停止時間 s1 は，調整後の処理時

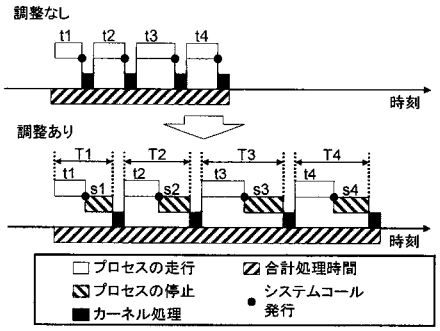

図1 プロセスの走行制御の様子

間と非調整の処理時間の差

$$s1 = 2*t1 - t1 = t1 \quad (2)$$

となる．

したがって，調整の程度を n%とした場合，停止時間を S(n)，プロセッサ使用量を t(n)とすると，

$$S(n) = (100/n)*t(n) - t(n) \quad (3)$$

となる．

2.3 プロセス停止方法

プロセスの停止方法は，大きく 2 つある．1 つはシステムクロックを監視して待つ方式(以降，監視方式と呼ぶ)であり，もう 1 つは休眠と覚醒により行う方式(以降，待ち方式と呼ぶ)である．

監視方式は，システムクロックの最小単位で停止時間を調整できるため，停止時間を細かく指定できる(要望 1)．しかし，停止処理中はプロセッサを使用するため，制御オーバヘッドを抑制(要求 2)することはできない．一方，待ち方式は監視方式と逆の特徴を持つ．つまり，細かい停止時間の指定はできないが，停止処理の間はプロセッサを使用しないため，制御のオーバヘッドを抑制(要求 2)できる．このため，プロセスの停止には，待ち方式を採用する．

本制御機構はライブラリ方式で実現(要求 1)するため，プロセスの停止にはシステムコールを用いる．待ち方式でプロセスを停止させるシステムコールとして，大きく 2 つある．1 つは wait 系システムコールであり，もう一つは sleep 系システムコールである．両者の比較を表 1 に示す．wait 系システムコールは，停止時間が指定できないうえに，再走行の契機は指定プロセスの状態変化に依存する．状態変化とは，プロセスの終了や停止などである．一方，sleep 系システムコールは，停止時間が指定可能であり，指定停止時間が経過したら再度プロセスを走行させる．また，例えば UNIX 系の nanosleep システムコールの場合，最小停止時間が 1 ナノ秒と短く，調整精度の向上(要望 1)が見込まれる．以上から，プロセスの停止には，sleep 系システムコールを使用する．

表1 プロセスを停止するシステムコールの比較

方式	wait 系システムコール	sleep 系システムコール
引数	指定プロセスの PID など	停止時間
停止時間	他プロセスに依存	指定可能
再走行の契機	指定プロセスの状態変化	指定時間の経過

2.4 調整の程度の指定法

自由な調整制御(要求 3)を満足するには，被調整プロセスと性能指定プロセスの間で調整の程度に関する情報を授受できる必要がある．具体的には，性能指定プロセスが指定した調整の程度に関する情報を，被調整プロセスが利用するライブラリ内で授受できる必要がある．これへの対処として，アドレス指定で直接に情報を設定や参照できることから，共有メモリを利用する．

共有メモリを利用した調整の程度の授受法を図 2 に示す．利用者は，性能調整を行う前に共有メモリを作成し，作成した共有メモリに調整の程度を設定する．実行速度の調整は，被調整プロセスがライブラリ内で共有メモリにアクセスし調整の程度を取得することで行う．システムコールの発行ごとに共有メモリの値を参照することで，実行途中にプログラム実行速度を変更できる．

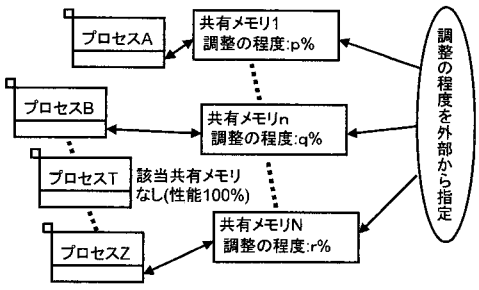

図2 調整の程度の授受法

2.5 制御処理の流れ

2.5.1 基本的な処理の流れ

2.1 節で示した(要求)と(要望)を満足するライブラリの制御処理の流れを図 3 に示し，以下に説明する．

(1)プロセッサのシステムカウンタを用いて現システムカウンタ値(以降，現時刻と呼ぶ)を取得する．

(2)共有メモリから，開始時刻(後述する)と調整の程度を取得する．

(3)プロセッサ使用量(現時刻－開始時刻)と調整の程度から，式(3)に基づき停止時間を算出する．

(4)停止処理を行う．(詳細については 2.5.2 項で述べる)

(5)AP に依頼されたシステムコール処理を呼び出す．

(6)プロセッサのシステムカウンタを用いて現システムカウンタ値(以降，開始時刻と呼ぶ)を取得し，共有メモリに保存する．

2.5.2 停止処理

停止処理では，sleep 系システムコールを用いてプロセスを停止させる．この場合，利用する sleep 系システムコールの最小停止時間(W)が問題になる．つまり，算出した停止時間(S)が W より小さい場合，制御ができない．

そこで，制御のために閾値(L)を設け，S≦L の場合はプロセスを停止させず，そのまま要求システムコールを発行する．ただし，この際の S は以降の契機ごとに加算し，(加算した S)＞L になったときにプロセスを停止させる．この処理の様子を図４に示し，以降に説明する．

(1)共有メモリから合計停止時間を取得する．

(2)合計停止時間と算出した停止時間(以降，現停止時間と略す)を加算し，合計停止時間とする．

(3)合計停止時間(S)と閾値(L)を比較する．

(A)S＜＝L の場合：S を合計停止時間として，共有メモリに保存する．

(B)S＞L の場合：sleep 系システムコールにより S 時間だけ停止し，合計停止時間を 0 として共有メモリに保存する．

上記の処理において，閾値(L)は sleep 系システムコールの最小停止時間(W)より大きいことが必要である．

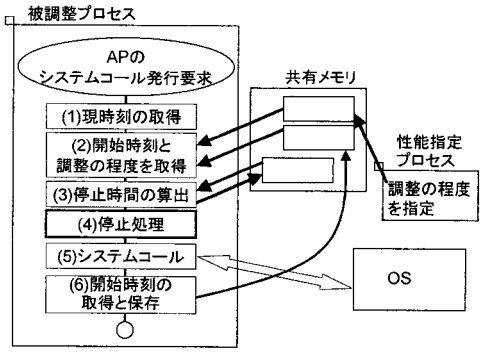

図 3 制御処理の流れ

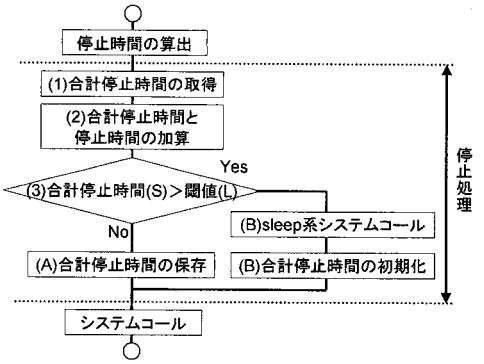

図 4 停止処理の流れ

3. 実装と評価

3.1 実装内容

2.2 節で述べたように，AP が依頼したシステムコール処理を OS に依頼する直前にこれまでに述べた機能を実装する．そこで，FreeBSD 4.3-RELEASE におけるシステムコール発行の流れを図 5 に示す．

プログラムを実行する際，そのプログラムを実行するプロセスはライブラリを読み込む．そのプロセスがシステムコール，またはライブラリコールを発行する場合，プロセスは読み込んだライブラリ内に展開されている処理を行い，ソフトウェア割込(int0x80 命令)を発行して OS に処理を渡す．OS では，対応するシステムコールの処理を行う．

本実装では，被調整プロセスを一時停止させるために，ソフトウェア割込を発行する直前に処理をフックし，sleep 系システムコールを用いてプロセッサ量を調整する機能を，既存のライブラリに追加する．

3.2 プロセス最小停止時間

停止処理に用いる sleep 系システムコールには，sleep ライブラリコール，usleep ライブラリコール，nanosleep システムコールの 3 つがあ

り，それぞれの最小停止時間(W)は 1 秒，1 マイクロ秒，1 ナノ秒である．(要望 1)を満たすため，プロセスの停止には nanosleep システムコールを利用する．しかし，sleep 系システムコールの仕様には，以下の問題点が存在する．

(問題点)sleep 系システムコールは少なくとも指定された時間必ず停止する．しかし，プロセスが再び実行できるようになるまで指定された時間より，最大 10 ミリ秒余計に時間がかかる．

このため，合計停止時間(S)と実際の停止時間には，最大 10 ミリ秒の差が生じる．そこで，プロセス停止処理を行う直前に，合計停止時間(S)から 10 ミリ秒の減算を行う．こうすることで，合計停止時間(S)と実際の停止時間の差は小さくなる．

しかし，閾値(L)を 10 ミリ秒以下とした場合，この方法では速度調整が行えない．このため，最小停止時間(W)を 10 ミリ秒とし，合計停止時間(S)＞閾値(L)となる場合は，10 ミリ秒減算した後に停止処理を行う．

3.3 評価環境と評価プログラム

Celeron(2.8GHz)プロセッサを搭載したマシンで FreeBSD 4.3-RELEASE(以降，FreeBSD とする)を走行させ，実装したライブラリを使ってプログラムの処理時間を測定した．他プロセスの影響を避けるため，FreeBSD をシングルユーザモードで起動した．また，共有メモリは 32 バイト分の領域を持つものを 1 つだけ作成した．

評価プログラムを図 6 に示す．評価プログラムとして，以下の 2 つを用意した．

プログラム A：CPU 処理(特定のメモリ領域の値のインクリメントを繰り返す処理)と getpid システムコール発行を繰り返し行うプログラム

プログラム B：CPU 処理と入力処理(DK の raw デバイスから 512byte 読み込む処理)を繰り返し行うプログラム

図 6(a)はプログラム A について示したものであり，図 6(b)はプログラム B について示したものである．

3.4 評価の観点

試作したライブラリを使用してプロセスを走行させるうえで，追加した処理が与える影響は小さいほうがよい．そこで，試作ライブラリのオーバヘッドを測定した．また，比較的精度のよい閾値(L)を求めるため，閾値(L)を変化させ評価を行った．

次に，さまざまなアプリケーションの性能調整を行う場合を考慮し，調整の限界，および調整の精度について評価を行った．調整の限界は，プロセス停止時間を小さくして調整できる限界を評価した．また，調整の精度では，処理内容の違うプロセスの性能を調整し評価を行った．さらに，共存プロセスが走行した場合に，共存プロセスが調整に与える影響について評価を行った．具体的には，共存プロセスの個数が与える影響，および共存プロセスの処理内容が与える影響について評価を行った．

なお，以降の各図の実測値は，500 回の平均処理時間である．また，処理時間差とは，実測値と理論値の差を理論値で割算した値[%]である．ここで，理論値とは，当該の調整の程度で理想的に性能調整を行ったと仮定して算出した処理時間である．処理時間差が正の数の場合，実測値が理論値よりも長時間で処理を行った場合である．一方，負の数の場合，実測値が理論値よりも短時間で処理を行った場合である．

3.5 評価考察

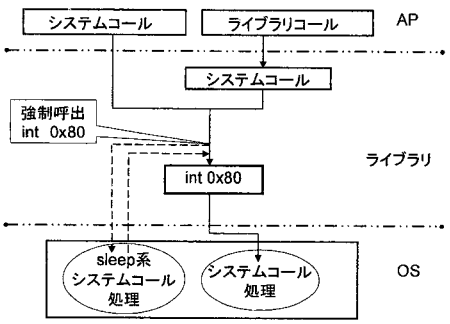

図 5 システムコール発行の流れ

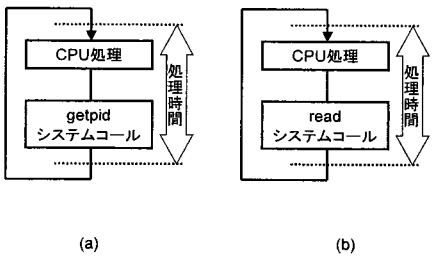

図 6 評価プログラム

3.5.1 追加処理のオーバヘッド

試作ライブラリに追加した処理がプロセスの走行に与える影響を明らかにするため，試作ライブラリのオーバヘッドを評価した．評価にはCPU処理時間を10ミリ秒にしたプログラムAを使用した．評価結果を表2に示す．

表2より，2つのライブラリでの処理時間の1回分の時間差平均は0.011ミリ秒であり，ライブラリの違いによる処理時間の差はほとんどないことがわかる．このため，試作ライブラリに追加した処理はプロセスの走行にほとんど影響を与えないといえる．

3.5.2 閾値による影響と調整の限界

次に，良い精度での性能調整を行うため，停止処理の閾値(L)について評価した．CPU処理を10ミリ秒にしたプログラムAを用いて，閾値(L)を10から50ミリ秒まで変化させ評価を行った．また，調整の程度を10%から90%まで変化させて評価した．測定結果を図7，図8および図9に示す．図7は閾値(L)を変化させた場合の1回の平均処理時間を示し，図8は閾値(L)を変化させた場合の処理時間差を示す．図9は調整の程度ごとの処理時間差の平均値を示す．各図より，以下のことがわかる．

(1)図7より，プログラム実行速度をあまり遅らせない場合(調整の程度 50%以上)でも，性能調整が行えることがわかる．このため，3.2節で述べたnanosleepシステムコールの問題点は調整に影響を与えない．

(2)図8と図9より，パラメータを10ミリ秒と30ミリ秒にした場合が比較的精度がよい．

これらのことから，閾値(L)を10ミリ秒，または30ミリ秒としたとき，精度のよい性能調整が期待できる．30ミリ秒とした場合，10ミリ秒とした場合に比べプロセス停止処理間の時間が長くなるため，きめの細かい調整が行えない．このため，閾値(L)を10ミリ秒にした場合がもっとも精度がよいといえる．

3.5.3 調整の精度

閾値(L)を10ミリ秒とした場合の調整の精度について明らかにするため，異なる2つのプログラムを用いて，調整の程度を10%から90%まで変化させ評価を行った．測定結果を図10と図11に示す．図10はプログラムAにおいてCPU処理の時間が100ミリ秒の場合であり，図11はプログラムBにおいてCPU処理の時間が100ミリ秒の場合である．

各図より，CPU処理時間が長いとき，実I/O処理の有無に関わらず，うまく性能調整を行えることがわかる．なお，このときの処理時間差は，それぞれ4%以下であった．

図2 ライブラリの違いによるオーバヘッド

既存ライブラリ	試作ライブラリ
10.018[ミリ秒]	10.029[ミリ秒]

3.5.4 共存プロセスが調整に与える影響

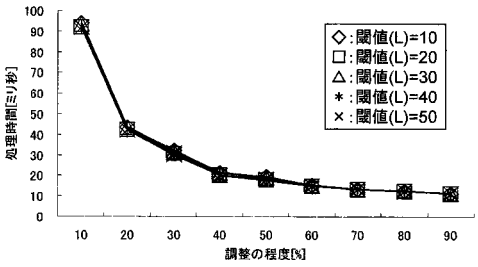

図7 平均処理時間

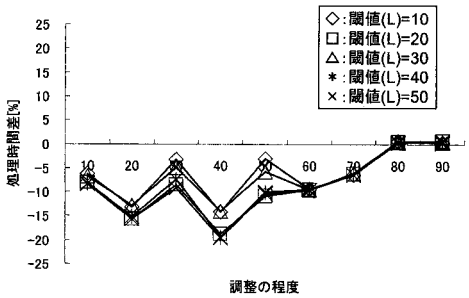

図8 処理時間差(調整の程度別)

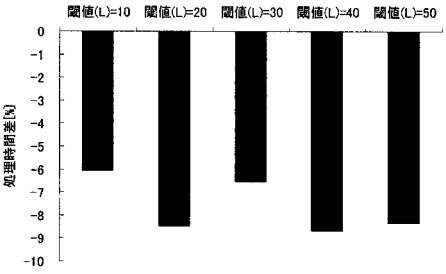

図9 処理時間差(パラメータごとの平均)

前項までの評価により，単一プロセスを性能調整する場合，十分な性能調整を行えることを示した．そこで，共存プロセスが存在する場合の調整への影響について評価した．評価結果を図 12 から図 15 に示す．

図 12 および図 13 はプログラム A において CPU 処理の時間が 100 ミリ秒の場合に，共存プロセス数を 0 から 5 個に変化させた場合である．共存プロセスには，図 12 は CPU 処理が 100 ミリ秒のプログラム A を使用し，図 13 は CPU 処理が 100 ミリ秒のプログラム B を使用した．また，図 14 および図 15 はプログラム B において CPU 処理の時間が 100 ミリ秒の場合に，共存プロセス数を 0 から 5 個に変化させた場合である． 共存プロセスには，図 14 は CPU 処理が 100 ミリ秒のプログラム A を使用し，図 15 は CPU 処理が 100 ミリ秒のプログラム B を使用した．

各図より，以下のことがわかる．

(1)プロセスの走行をあまり遅らせない場合(調整の程度 60%以上)は，共存プロセスの処理内容に関わらず，被調整プロセスは共存プロセスの影響を大きく受けることがわかる．また，共存プロセス数が多くなるほどその影響は大きい．例えば，調整の程度を 90%にした場合，処理時間は，共存プロセス数が 1 つなら約 100%，5 つな

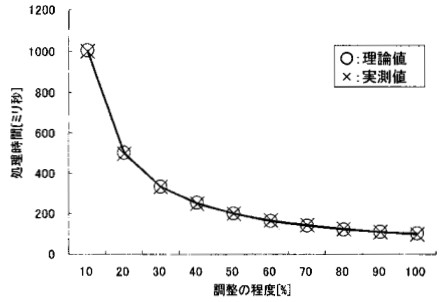

図 10 getpid 処理(CPU 処理 100 ミリ秒)

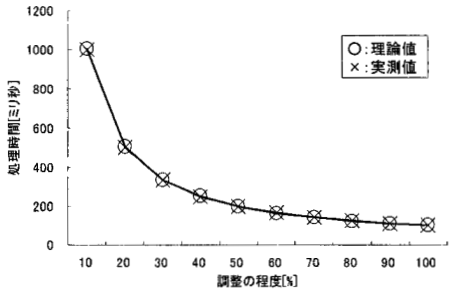

図 11 DK の read 処理(CPU 処理 100 ミリ秒)

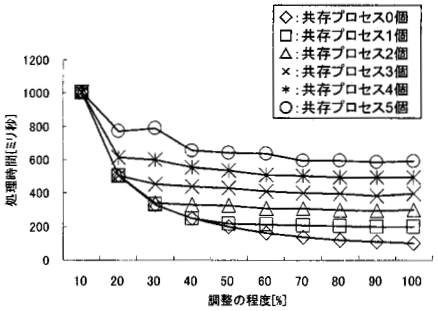

図 12 getpid 処理(共存プロセス：getpid 処理)

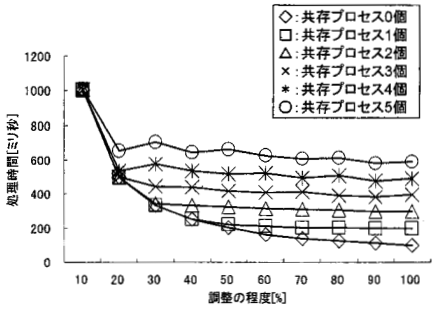

図 13 getpid 処理(共存プロセス：read 処理)

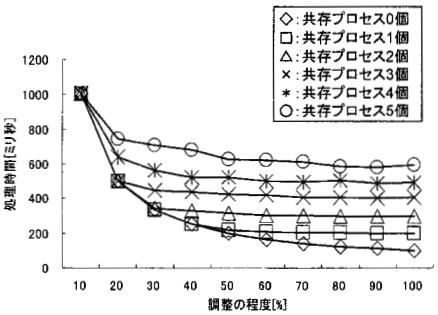

図 14 read 処理(共存プロセス：getpid 処理)

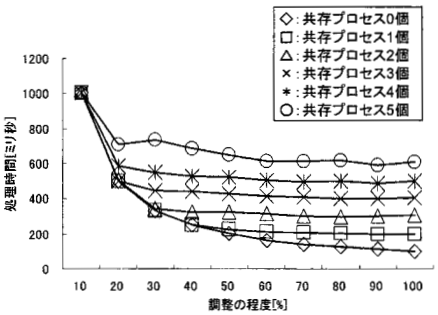

図 15 read 処理(共存プロセス：read 処理)

ら約 500%増加する．これは，プロセッサ使用量の決定方法に起因する．本方式では，AP がシステムコール発行から次のシステムコール発行まで連続でプロセッサを使用すると仮定し，その時間をプロセッサ使用量としている．この方法では，共存プロセスが走行している場合，タイムスライスやプリエンプションで被調整プロセスが実行されていない時間もプロセッサ使用量に含まれてしまう．このため，ライブラリ方式での実装では，OS 方式での実装よりも他プロセスの影響を受ける．

(2)プロセスの走行を大きく遅らせる場合(調整の程度 40%以下)は，共存プロセスの影響を受けない場合があることがわかる．具体的には，共存プロセス数が 1 つ，2 つ，および 3 つの場合，被調整プロセスの調整の程度がそれぞれ 40%, 30%, 20%以下のとき共存プロセスの影響を受けないことがわかる．また，調整の程度が 10%の場合，共存プロセスの個数に関係なく，共存プロセスの影響を受けないことがわかる．これは，FreeBSD のスケジューラが走行時間の短いプロセスに対し優先度を高く設定し，優先的に走行させるためである．

4. おわりに

プロセッサ使用量の制御によるプログラム実行速度調整法について述べた．システムコールの発行間隔に着目して，プロセッサ使用量を制御しプログラム実行速度を調整するライブラリを提案し，評価した．プロセスの停止法として，wait システムコールと nanosleep システムコールがあるが，最小停止時間の小ささと停止時間を指定可能であることから，プロセスの停止法には nanosleep システムコールを採用した．計算機の使用者が調整の程度を任意に決定できるように，共有メモリを使用し調整の程度を指定する．共有メモリを使用することで，上記の利点に加え，被調整プロセスと共存プロセスを分離し任意のプロセスの性能調整を可能にした．また，プロセス停止処理において閾値(L)を設け，合計停止時間が L より大きい場合のみプロセスを停止する仕様とした．

さらに，試作したライブラリを評価した．既存のライブラリに追加した処理の影響は非常に小さく，広い範囲での性能調整が可能であることを示し，閾値(L)が 10 ミリ秒の場合に調整の精度がもっともよいことを示した．また，単一プロセスのみの性能調整では，実 I/O 処理の有無やシステムコール発行間隔に関わらず，調整の精度が十分であることを示した．一方，共存プロセスが存在した場合，被調整プロセスは共存プロセスの影響を大きく受けることを示した．これは，プロセッサ使用量を決定する際，AP がシステムコール間にプロセッサ連続して使用していると仮定したことに起因する．

残された課題として，共存プロセスが存在する場合の制御方式の確立がある．

謝辞 本研究の一部は，科学研究費補助金基盤研究(B)(18300010)，および科学研究費補助金若手研究(B)（課題番号 18700030）による．

参考文献

[1] 谷口秀夫，坂口修，“入出力回数の制御によりサービス時間を調整する制御法，” 信学論(D-I)，vol. J81-D-I，no. 11，pp. 1211-1218，Nov. 1998.
[2] 谷口秀夫，“入出力時間の制御によりプログラム実行速度を調整する制御法，” 信学論(D-I)，vol. J83-D-I，no. 5，pp. 469-477，May. 2000.
[3] 谷口秀夫，“入出力性能の制御によりプログラム実行速度を調整する制御法の実装方式による比較評価，” 信学論(D-I)，vol. J84-D-I，no. 9，pp. 1362-1371，Sep. 2001
[4] 谷口秀夫，“サービス処理時間を調整するプロセスのスケジュール法，” 信学論(D-I)，vol. J81-D-I，no. 4，pp. 386-392，1998.
[5] 谷口秀夫，“プロセススケジュールの制御によるプログラム実行速度調整法の評価，” 信学論(D-I)，vol. J83-D-I，no. 1，pp. 184-193，Jan. 2000.
[6] 田端利宏，谷口秀夫，“複数サービスの処理時間を調整するプロセススケジュール法，” 信学論(D-I)，vol. J86-D-I，no. 7，pp. 458-468，Sep. 2003.
[7] Anna Povzner，Tim Kaldewey，Scott Brandt，Richard Golding，Theodore M. Wong，Carlos Maltzahn，“Efficient Guaranteed Disk Request Scheduling with Fahrrad，” EuroSys'08，pp. 13-25，Apr. 2008
[8] H. P. Chang，R. I. Chang，W. K. Shih，R. C. Chang “GSR: A global seek-optimizing real-time disk-scheduling algorithm，” The Journal of Systems and Software，pp. 198-215，2007
[9] S. A. Brandt，S. Banachowski，C. Lin，and T. Bisson. “Dynamic integrated scheduling of hard real-time，soft real-time and non-real-time processes，” RTSS2003，pp. 396-407，Dec. 2003
[10] M. B. Jones，D. Rosu，and M.-C. Rosu，“CPU reservations and time constraints: Efficient，predictable scheduling of independent activities” SOSP，pp. 198-211，Oct. 1997