システムソフトウェアと
オペレーティング・システム 85－13
（２０００．８．４）

入出力性能の制御による プログラム実行速度調整制御法の評価

谷口 秀夫
九州大学　大学院システム情報科学研究院

計算機ハードウェアの性能向上にともない，ソフトウェアの処理時間は短縮し，複雑な処理の実行が可能になっている．一方，ソフトウェアの処理速度は，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，利用者が求める速度あるいはソフトウェアが提供するサービス内容に合わせた速度で処理を実行する制御方式の確立が必要である．本論文では，入出力の性能を制御し，プログラムの実行速度を調整する制御法の実装方式を説明し比較している．具体的には，入出力の回数を調整する基本方式，入出力の時間を調整する基本方式を説明し実装評価している．また，入出力の時間を調整する方式をライブラリとして実装し，OSカーネル実装と比較して，その特徴を明らかにしている．

Evaluation of Mechanisms for Regulating Program Execution Speed based by Controlling I/O Performance

Hideo TANIGUCHI
Graduate School of Information Science and Electrical Engineering, Kyushu University

Improvement of computer hardware performance cuts down the time of software processing, as a result it is possible for software to execute complex processing. By the way, improvement of computer hardware performance influences the time of software processing. So some service programs are coded to obtain the suitable time of service processing. If program execution speed can be regulated, we can control the program execution speed with our favorite. I suggested the mechanisms that regulate program execution speed by controlling process schedule or controlling I/O performance. In this paper, I describe two implement methods (internal kernel implement and external kernel implement) for the mechanism that regulates program execution speed by controlling I/O performance. This paper describes the basic mechanism, and shows two implement methods. And it reports a result of implementation and evaluation.

1. まえがき

最近，計算機ハードウェアの性能向上が著しいため，低性能から高性能まで様々な性能を持つ計算機が同時に存在している．一方，ソフトウェアの処理速度はハードウェア性能に大きく依存する．このため，例えば，表示速度が計算機毎に大きく異なってしまい，利便性が低下することがある．さらに，近年，計算機を利用したソフトウェア教材が開発されつつある．ソフトウェア教材は，学習者が内容を理解するためにアニメーションなどで説明を行っている．この場合，学習者はその説明を何度も，かつ，自分が納得できる速度で見たいという要望がある．一方，ゆっくり動けばよいというだけではなく，アルゴリズムの教材ではその効率を体感するために高速で動いてほしいときもある．つまり，このようなソフトウェアでは，利用者が実行速度を自分の目的に合わせて変更したいことが多い．また，プログラム実行速度を実行中に動的に変更して，飛ばし見したい場合もあるかもし

れない．つまり，利用者の要求に合わせた速度にサービス処理速度を自由に調整したい要望がある．この機能を各ソフトウェア教材プログラムに組み込むことは，非常に大変である．

そこで，筆者は，計算機ハードウェアの性能の範囲で，利用者が求める速度でプログラムの実行速度を自由に調整できる方法について，研究を進めている．現在までに，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する方法[1]-[3]や，プロセスの入出力性能を調整する方法[4][5]を提案してきた．

本稿では，入出力性能を調整する方法をライブラリとして実装評価し，オペレーティングシステム（以降、OSと略す）カーネル実装と比較して，その特徴を示す．具体的には，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として，入出力時間を調整する方法と入出力回数を調整する方法を説明する．次に，入出力時間を調整する方法として優れた方式（wait方式）と入出力回数を調整する方法として優れた方式（周期固定方式）をOSカーネルに実装し評価する．さらに，wait方式をライブラリ実装し，実装方式による特徴比較を，性能調整の精度，性能調整対象でない他プロセスの影響，および性能調整する入出力の時間や利用する計算機のプロセッサ性能が性能調整の精度に与える影響の観点で行う．

2. 入出力性能の調整制御法の比較

2.1 入出力性能の調整制御法

2.1.1 基本方式

入出力の性能を調整する制御法[4][5]は，大きく以下の二つに分類できる．一つは，入出力時間を調整する方法である．これは，一つの入出力に要する時間を調整するもので，具体的には，入出力の終了時期を調整する．もう一つは，入出力回数を調整する方法である．これは，単位時間における入出力の回数を調整するもので，具体的には，入出力の開始時期を調整する．

各方式を図1に示す．図1(A)は入出力時間を調整する方法を示したものであり，実I/O時間に待ち時間を加え要求された入出力時間としている．図1(B)は入出力回数を調整する方法であり，単位時間内の入出力回数を要求された入出力回数に合わせるため，入出力を遅延して実行している．

入出力時間を調整する方法は，各入出力毎にその時間を調整できるため，きめ細かな入出力性能の調整が可能である．しかし，頻繁に入出力が発生する場合には，入出力の衝突により調整できない可能性が高くなる．一方，入出力回数を調整する方法は，頻繁に入出力が発生する場合でも単位時間の大きさを大きくすることにより，入出力性能の調整が可能である．しかし，この方法は，調整のきめ細かさが単位時間の大きさに左右され，入出力時間を調整する方法に比べ，きめ細かな入出力性能の調整はできない．

2.1.2 入出力時間の調整制御法

入出力時間を調整する方法では，入出力を行うOS処理と装置の実入出力（以降，裸入出力と名づける）の時間の扱いが問題になる．裸入出力の時間は，各入出力装置毎にハードウェア性能に合わせ事前に時間を決定しておく方法（以降，予測法と名づける）と，入出力要求の度に時間を測定する方法（以降，実測法と名づける）がある．予測法は，処理は簡単であるが，入出力するデータ長の変化を考慮することは難しく，またI/Oバッファのように OSが行う入出力の効率化策による入出力時間の短縮を反映できない欠点がある．このため，実測法を利用する．

入出力時間の調整処理は，開始処理，裸入出

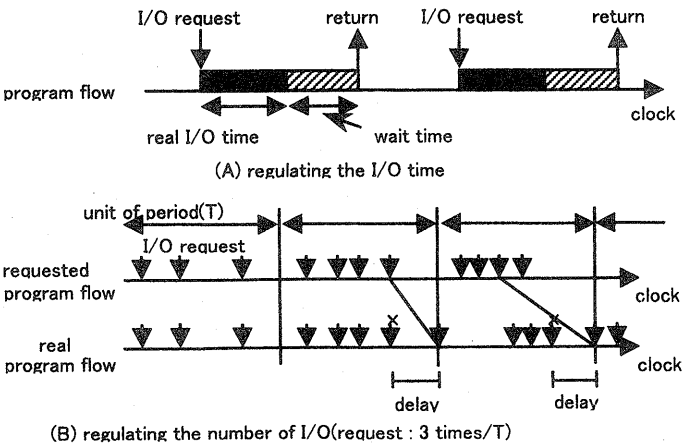


図1 入出力性能を調整する制御法

力処理，および終了処理からなる．裸入出力処理はOSカーネルの入出力処理であり，開始処理と終了処理が入出力時間を調整するための処理である．開始処理では，入出力要求を受け付け，裸入出力中であれば待ち，裸入出力中でなければ裸入出力処理の開始時刻を取得する．終了処理は，裸入出力処理の終了時刻を取得し，裸入出力の時間を算出し，要求性能に合わせた遅延処理を行った後，処理を終了する．このとき，裸入出力待ちのプロセスが存在すれば，待ちにある先頭のプロセスを起こす．要求性能がn倍の場合，要求性能に合わせた遅延処理では，実測した裸入出力処理時間の(n-1)倍の時間だけの遅延処理を行う．この遅延処理の方式には，大きく二つある．一つは遅延をハードウェアが持つシステムクロックの監視により行う方式（以降，loop方式と名づける）であり，もう一つは遅延をプロセスの休眠と覚醒により行う方式（以降，wait方式と名づける）である．

loop方式は，システムクロックの最小単位で遅延を調整できるため，精度の高い入出力時間の調整が可能である．しかし，遅延処理の間はプロセッサを占有するため，他のプロセスへの影響が大きい欠点がある．一方，wait方式は，loop方式とは逆の特徴を持つ．つまり，OSカーネルが提供する時計機能により休眠と覚醒を行うため精度の高い入出力時間の調整はできないものの，遅延処理の間は休眠しているため他のプロセスへの影響は小さい．

2.1.3 入出力回数の調整制御法

入出力回数を調整する方法は，単位時間の扱いにより，二つの方式がある．一つは，単位時間をシステムで一意とし，全てのプロセスを同じ単位時間の周期で扱う方式（以降，周期固定方式と呼ぶ）である．もう一つは，単位時間はシステムで一意ながらも，各プロセス毎に単位時間の周期を設定する方式（以降，周期可変方式と呼ぶ）である．さらに，単位時間をプロセス毎に設定する方式も考えられるが，この方式は，単位時間の違いを入出力回数で調整することにより前記の方式と同様になる．

二つの方式をシミュレーションにより評価し，周期可変方式に比べ，実Ｉ／Ｏがネックでない場合，周期固定方式は，要求された入出力性能で同時走行する多くのプロセスを制御できることを示した[4]．

2.2 比較

入出力時間を調整する方法として，wait方式は，他のプロセスへの影響が小さい[5]．また，入出力回数を調整する方法として，周期固定方式は，要求された入出力性能で同時走行する多くのプロセスを制御できる[4]．そこで，wait方式と周期固定方式を比較する．

両方式をBSD/UNIXのOSカーネルに実装した．評価として，プロセッサPentium90MHzの計算機を利用して，IDE磁気ディスク(800Mbyte)を使って行なった入力性能の調整機能について実測した内容を述べる．測定環境を以下に示す．

（1）余計なディスクアクセスを避けるためにシングルユーザモードで測定した．

（2）評価プログラムの中の入力操作は，入出力バッファのキャッシュヒットの影響をなくすためにディスクをrawデバイスとしてアクセスし，ディスクシーク時間の影響をなくすためにランダムなセクタから読み込むようにした．

（3）単位時間は1秒とした．

なお，評価プログラムは，read()システムコールにより512byteのデータ入力を繰り返し，入力の度にランダムなバイト数だけlseek()システムコールでアクセス位置をずらすものである．

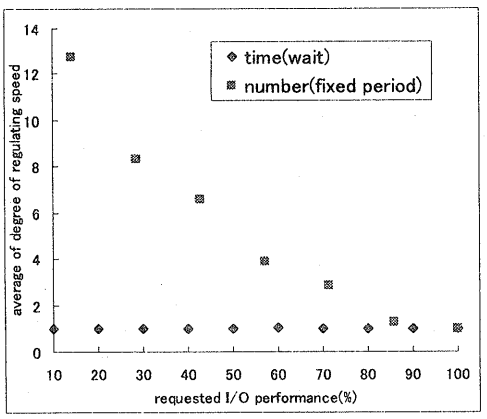
測定の結果として，速度調整度[3]の平均と分散を図2に示す．なお，周期固定方式では，ハードウェアが提供する最大の性能（生性能）における入出力回数を100%としている．図2からわかるように，速度調整度の平均と分散は共にwait方式が小さい．このため，wait方式は優れた入出力性能の調整方式といえる．

3. 実装方式の比較

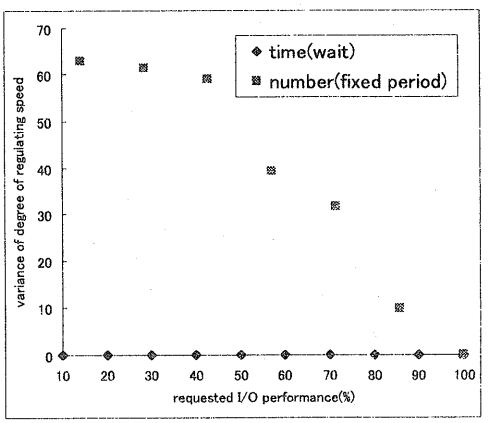
前章では，プログラム実行速度を調整する制御法として，入出力回数に比べ入出力時間を調整する方式が良いことを示した．そこで，本章では，入出力時間を調整するwait方式について，その実装方式による特徴を比較する．

3.1 OS方式とライブラリ方式

プログラム実行速度を調整する制御法を実装する方式として，大きく二つある．OSカーネル内へ実装する方式（OS方式）とOSカーネル外にライブラリとして実装する方式（ライブラリ方式）である．一般に機能を実装する場合，OS方式とライブラリ方式では表1に示す違いがある．



(A) average



(B) variance

図2 wait方式と周期固定方式の比較

表1 実装方式の比較

方式	OS方式	ライブラリ方式
長所	(1) 処理が速い(簡単) (2) APの実行ファイルを そのまま利用可	(1) OSの変更なし (多くの計算機で利用可)
短所	(1) OSの変更あり (OS作成環境が必要) (実行時に当該OSを走行)	(1) 処理が遅い(複雑) (2) 実現機能に制約 (3) APの再コンパイル要

OS 方式は，OS カーネルを変更するために様々な短所があるものの，処理が速く，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利用できる長所がある．一方，ライブラリ方式の特徴は，OS 方式の逆である．しかし，ライブラリ方式の長所である「OS の変更なし」は，OS 作成環境が不用であり，かつ利用できる計算機を広める意味で重要である．

入出力時間を調整する機能を実装する場合，上記の比較に加え，実装した機能が提供する調整の精度の比較が必要である．OS 方式では他の処理の影響を受けにくいが，ライブラリ方式では，他の処理の影響を受け易いので，調整の精度が低くなると推察できる．

3.2 実現内容

OS 方式とライブラリ方式を BSD/UNIX カーネル内およびカーネル外に実現した．

各方式の調整処理の流れを図 3 に示す．文献[5]に示したように，調整処理は，開始処理，裸入出力処理，および遅延処理を含む終了処理からなる．図 3 (A) に示したように，OS 方式では，これらの処理は OS カーネル内で行われ，ライブラリの処理はない．一方，図 3 (B) に示したように，ライブラリ方式では，これらの処理はライブラリで行われるため，その実現に必要となるシステムコールが 6 回発行される．シグナル登録は signal，時刻取得は gettimeofday，入出力は read または write，休眠と覚醒は usleep（システムコールとしては setitimer と select），のシステムコールである．なお，ライブラリ方式は，遅延処理の違いから二つの場合を実現した．一つは遅延時間の値に関係なく無条件に関数 usleep を呼び出す場合（以降の各図では Lib.(A)と略す）であり，もう一つは遅延時間が 10 ミリ秒以上のときに関数 usleep を呼び出す場合（以降の各図では Lib.(B)と略す）である．これは，次の理由による．本来，usleep 機能はマイクロ秒単位での休眠と覚醒の機能であるので，場合 Lib.(A)の実装が正しい．しかし，BSD/UNIX の usleep では，休眠と覚醒の精度が 10 ミリ秒程度であり，かつ休眠時間が切り上げられる．このため，OS 方式の実装[5]と同様な遅延処理内容となる場合 Lib.(B)も実装した．

3.3 評価

3.3.1 測定環境と評価項目

測定は，Pentium(233MHz)プロセッサを搭載した計算機を用い，評価プログラムの処理時間の測定は，最小単位 10 ミリ秒の時間計測システムコールを利用した．また，裸入出力時間の測定は，OS 方式はカーネル内部の時間計測機能（最小単位 1 マイクロ秒）を利用し，ライブラリ方

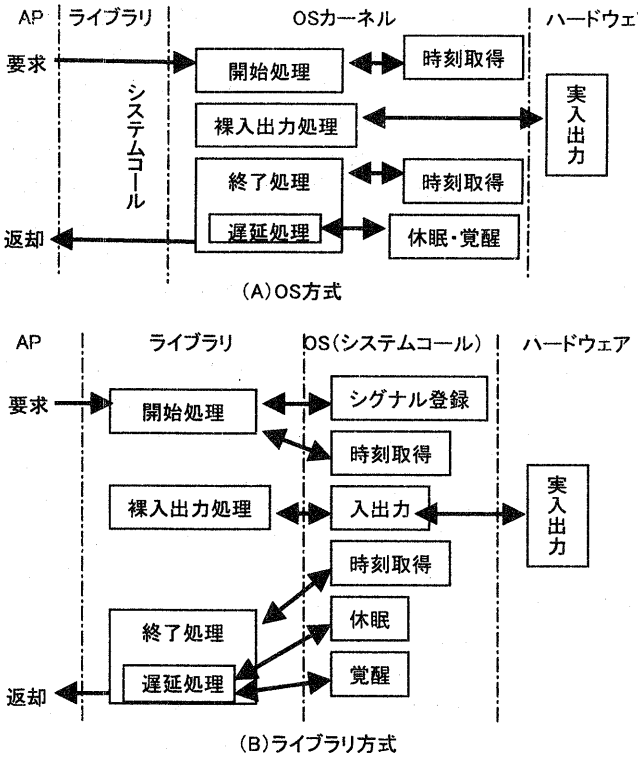


図3 調整処理の流れ

式は gettimeofday システムコールを利用した．

文献[5]では，OS 方式について，裸入出力の性能の範囲で，プロセス個別に自由な入出力時間の調整ができることが確認されている．同様に，ここでも，ライブラリ方式についても同様の確認ができた．そこで，OS 方式とライブラリ方式の調整の精度を評価した．具体的には，入出力時間が比較的長い磁気ディスク装置（以降，DK と略す）からのデータ入力および入出力時間が比較的短いコンソールへのデータ出力について，OS 方式とライブラリ方式を評価した．また，入出力時間の性能調整を行う際には，入出力時間の性能を調整されるプロセス以外にも他のプロセスが走行していることが十分想定される．そこで，他にプロセスの処理が入出力時間の性能調整に与える影響を評価した．次に，様々なプロセッサ性能を持つ計算機が同時に存在する現状や将来を考慮し，ライブラリ方式について，入出力時間および遅延処理方式が性能調整に与える影響を評価した．

なお，特に説明のない限り，以降の各図の処理時間は，性能調整の処理をまったく行わない裸入出力時間の実測値を 1 とした相対値である．また，処理時間差とは，実測の処理時間と予想の処理時間の差を予想の処理時間で割算した値（%）である．ここで，予想の処理時間とは，当該の要求入出力性能の処理を行った時の裸入出力時間を要求入出力性能に合わせて理想的な性能調整を行ったと仮定して算出した処理時間である．

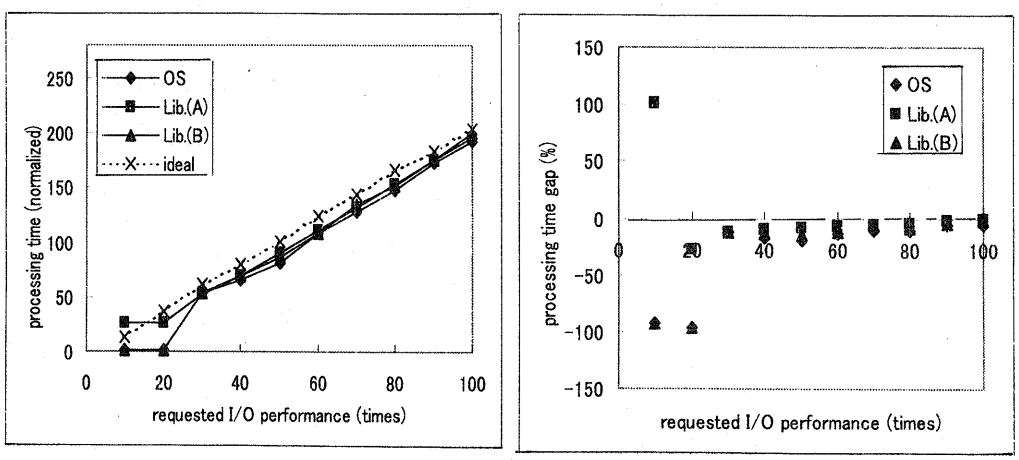


図4 OS方式とライブラリ方式の比較(磁気ディスクからのデータ入力)

3.3.2 OS方式とライブラリ方式の比較

＜調整の精度＞

調整の精度を明らかにするため，DKからのデータ入力および入出力時間が比較的短いコンソールへのデータ出力について評価した．

DKからのデータ入力では，評価プログラムはDKからのデータ入力を1024バイト単位で1000回行うもので，OS機能(I/Oバッファやデータ先読み)の影響を避けるため，特殊ファイルからの入力とした．要求入出力性能が10倍から100倍の場合について，処理時間を図4に示す．図4から以下のことがわかる．

（1）遅延処理内容が同様なOS方式とLib.(B)方式は，処理時間と処理時間差が同様であり，OS方式とライブラリ方式といった実装方式の違いによる調整の精度の違いはない．

（2）Lib.(A)方式は，要求入出力性能が10倍と20倍のとき，他の方式と処理時間や処理時間差が異なる．これは，usleep関数において，休眠と覚醒の精度が10ミリ秒程度であり，かつ休眠時間が切り上げられるためである．

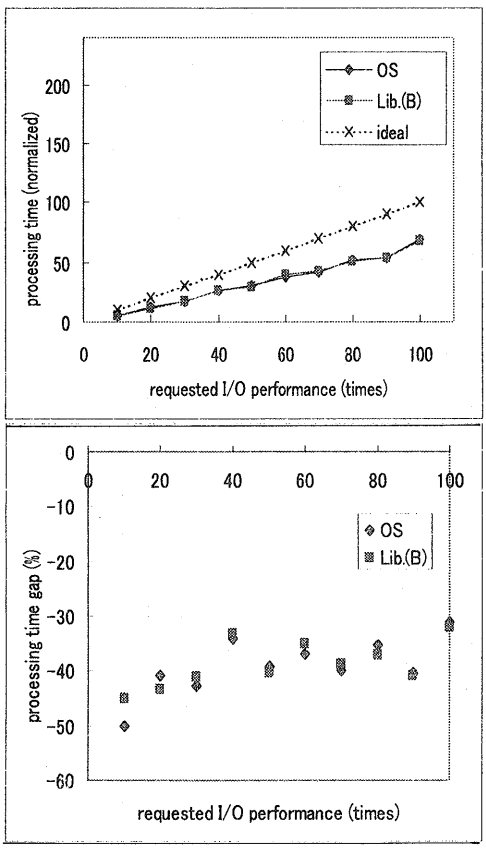


図5 OS方式とライブラリ方式の比較
（コンソールへのデータ出力）

コンソールへのデータ出力では，評価プログラムはコンソールへのデータ出力を1バイト単位で1000回行うものである．要求入出力性能が10倍から100倍の場合について，処理時間と処理時間差を図5に示す．図5から，遅延処理内容が同様なOS方式とLib.(B)方式は，処理時間と処理時間差が同様であり，実装方式の違いによる調整の精度の違いはないことがわかる．また，バッファへの複写処理への影響[5]を明らかにするため，要求入出力性能が数倍の場合の性能調整時間を測定した結果を図6に示す．図6より，ライブラリ方式でも，バッファへの複写処理の影響をOS方式と同様に受けることがわかった．つまり，実装方式に関係なく，出力データの単位が大きいほど傾き1の直線に近い理想的な性能調整ができる．

＜他プロセスの影響＞

他プロセスへ与える影響と他プロセスから受ける影響を明らかにするため，性能を調整されたプロセスとそうでないプロセス（他プロセス）を共存して走行させ，相互の影響を評価した．

他プロセスの走行が調整の精度に与える影響を明らかにする．上記と同様にDKからのデータ入力を1024バイト単位で1000回行う評価プログラムを利用し，他プロセスを走行させた状態で，処理時間を測定した．他プロセスとして二つ用意した．一つは，特定の変数の1加算を無限回繰り返す処理である．もう一つは，特定の変数の1加算を約5ミリ秒間繰り返す処理と約10ミリ秒のWAIT状態を無限回繰り返す処理である．要求入出力性能と処理時間の関係を図7に示す．図7から以下のことがわかる．

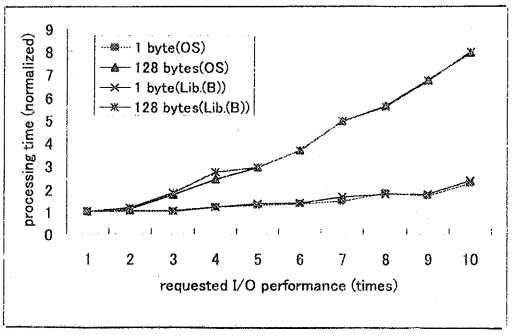


図6 要求性能と処理時間(コンソールへのデータ出力)

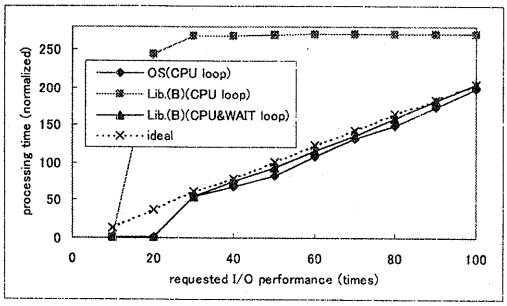


図7 他プロセスから受ける影響
（磁気ディスクからのデータ入力）

（1）特定の変数の 1 加算を無限回繰り返す処理プロセスと共存した場合，ライブラリ方式は，性能調整できない要求入出力性能10倍を除き処理時間が非常に大きくなっている．これは，性能調整処理内でのシステムコールからの戻りが他プロセスのタイムスライス契機に依存するためである．

（2）特定の変数の 1 加算を約 5 ミリ秒間繰り返す処理と約 10 ミリ秒の WAIT 状態を無限回繰り返す処理プロセスと共存した場合，ライブラリ方式は OS 方式と同様な処理時間となり，二つの方式は同様な性能調整ができるといえる．

したがって，プロセッサ使用率が高くない場合，ライブラリ方式は OS 方式と同様な性能調整ができるといえる．利用者とのインタラクティブな操作を主に行う計算機では，プロセッサ使用率はあまり高くないことが多いため，ライブラリ方式でも性能調整が可能である．

次に，性能を調整されたプロセスが他プロセスに与える影響を明らかにする．DK からのデータ入力を 1024 バイト単位で無限回繰り返す被調整プログラムを起動した後，特定の変数の 1 加算を繰り返す処理（他プロセス）の処理時間を測定した．測定結果を図 8 に示す．処理時間は，

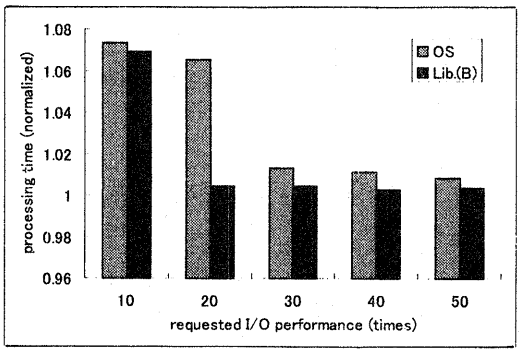


図8 他プロセスが受ける影響
（磁気ディスクからのデータ入力プロセスと共存）

他プロセスを単独で走行させたときの処理時間を 1 として正規化したものである．図 8 から以下のことがわかる．

（1）いずれの方式も他プロセスの処理時間の増加はせいぜい 8 %以下であり，他プロセスに与える影響は大きくない．

（2）OS 方式に比べ，ライブラリ方式は他プロセスに与える影響が少ない．これは，ライブラリ方式の場合，性能調整処理が OS カーネル外で行われるためである．OS 方式では，性能調整処理を OS カーネル内で行うため，他プロセスに影響を与えやすい．このため，要求入出力性能を小さくするとライブラリ方式よりも早期に大きな影響を与える．このことは，要求入出力性能 20 倍の場合に顕著である．

3.3.3 入出力時間と調整精度

ライブラリ方式について，入出力時間および遅延処理方式が性能調整に与える影響を明らかにする．このため，1 回の入出力時間が異なる DK からのデータ入力とコンソールへのデータ出力について，比較する．具体的には，DK からのデータ入力（図 4）に加え，コンソールへのデータ出力（1 バイト）についても Lib.(A)方式と Lib.(B)方式による処理時間を測定した．また，計算機のプロセッサの性能を変えて測定した．具体的には，コンソールへのデータ出力（1 バイト）について Lib.(A)方式と Lib.(B)方式による処理時間の測定をプロセッサ Pentium800MHz の計算機でも行った．結果を図 9 に示す．図 4 と図 9 から以下のことがわかる．

（1）プロセッサが Pentium233MHz の場合，図 5 では，要求入出力性能 10 倍と 20 倍では Lib.(A)方式と Lib.(B)方式とも性能調整が十分でなく，どちらも要求入出力性能 30 倍付近から性能調整が可能になっている．これに対し，図 9（Pentium233MHz）では，要求入出力性能 100 倍と 200 倍では Lib.(A)方式と Lib.(B)方式とも性能調整が十分でなく，どちらも要求入出力性能 300 倍付近から性能調整が可能になっている．要求入出力性能が小さいとき性能を調整できない理由は，タイマの精度が 10 ミリ秒であるまで 10 ミリ秒未満の遅延処理は行えないこと[5]による．したがって，入出力時間が長いほど要求入出力性能が小さくても性能調整が可能といえる．また，遅延処理方式の違いは要求性能

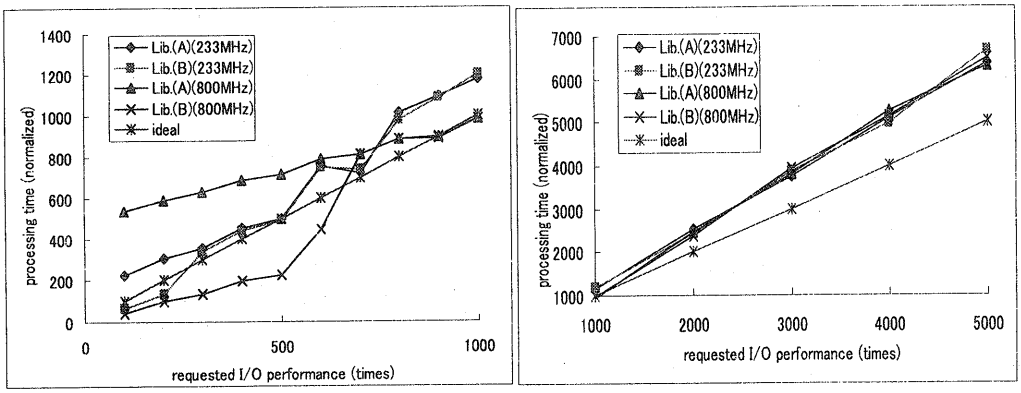


図9 プロセッサ性能の影響(コンソールへのデータ出力)

が小さいときに影響を与えるといえる.

(2) 図9より, プロセッサ性能が高く (Pentium800MHz) なると, Lib. (A)方式とLib.(B)方式とも性能調整が十分でない要求入出力性能の範囲が拡大し, 性能調整が可能になる要求入出力性能の値が大きくなる. これは, コンソールへのデータ出力ではプロセッサ処理の割合が多く, プロセッサ性能が高くなると入出力時間が短くなるためである.

(3) 図9より, 要求入出力性能が大きくなると, 入出力時間および遅延処理方式に関係なく, 性能調整が可能といえる.

4. むすび

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として, 入出力時間を調整する方法と入出力回数を調整する方法を説明した. 入出力時間を調整する方法として, 遅延処理をプロセスの休眠と覚醒により行うwait方式は, 他のプロセスへの影響が少ない方式である. 入出力回数を調整する方法として, 入出力回数を制限する単位時間を全てのプロセスで同じ単位時間の周期で扱う周期固定方式は, 同時走行する多くのプロセスの制御が可能な方式である. wait 方式と周期固定方式を OS カーネル内に実装し評価した結果, wait 方式は, 速度調整度の平均と分散が小さく, 優れた方式であること明らかにした.

また, wait 方式を OS カーネル外にライブラリとしても実装し, 実装方式による特徴比較を行い, 次のことがわかった. ①性能調整の精度は, 実装方式にあまり関係しない. ②他プロセスのプロセッサ使用率が高い場合, ライブラリ実装方式は, OS 実装方式より他プロセスの影響を受け易い. ③性能を調整されたプロセスが他プロセスに与える影響は, 両実装方式ともせいぜい8%以下である. なお, ライブラリ実装方式は, OS 実装方式に比べ他プロセスに与える影響が少ない. ④入出力時間が長いほど要求入出力性能が小さくても性能調整が可能である. したがって, 高性能ハードウェアでは入出力時間が短かくなるため, 要求入出力性能が大きくないと性能調整がうまく行かない. しかし, 利用者インタフェース上, ハードウェア性能の違いに関係なく同様なプログラム実行速度で利用したい場合, 低性能ハードウェアでは要求入出力性能が小さく, 高性能ハードウェアでは要求入出力性能が大きくなるため, この点は問題ではない.

残された課題として, 複数プロセスの入出力性能調整機能の検討がある.

<文献>

[1] 谷口秀夫: "可変なプロセッサ性能を提供するスケジュール法", 情報処理学会コンピュータシステムシンポジウム, シンポジウム論文集, Vol.94, No.10, pp.63-70 (1994).

[2] 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, 第 J81-D-I 巻, 第4号, pp.386-392 (1998).

[3] 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の実行速度調整法の評価", 信学論 D-I, 第 J83-D-I 巻, 第1号, pp.184-193 (2000).

[4] 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, 第 J81-D-I 巻, 第11号, pp.1211-1218 (1998).

[5] 谷口秀夫: "入出力時間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, 第 J83-D-I 巻, 第5号, pp.469-477 (2000).