Tender オペレーティングシステムにおける資源「演算」を用いたサービス処理時間の保証

田端利宏† 谷口秀夫†

Tender では，プロセスから，プロセッサを利用できる程度を資源として分離・独立化し，「演算」と名付けた．資源「演算」には，プログラムの実行性能を調整するものと優先度を持つものの 2 種類がある．2 種類の「演算」を複数個組み合わせてプロセスに関連付けることより，サービス処理時間の保証，ユーザレベルスケジューリング，および優先度を上げずにプロセッサの利用時間を増やすことが可能となる．本論文では，複数の資源「演算」をプロセスと関連付ける機構について述べ，複数の資源「演算」の利用法を提案する．また，複数の資源「演算」を利用したときのプログラムの実行速度調整機能の精度と処理の均一性，複数の優先度の演算を関連付ける効果，およびサービス処理時間保証の評価とサービス処理時間を保証するプロセスが他プロセスに与える影響についての評価結果を報告する．

Guarantee of Service Processing Time by *Execution* on *Tender* Operating System

Toshihiro Tabata† and Hideo Taniguchi†

The objects to be controlled and managed by operating system are called *resources* on ***Tender***. Resource *execution* has rate of processor utilization. There are two kinds of *execution*. One is to control speed of program execution. Another is scheduling processes by priority. To guarantee lowest performance of service is realized by attaching plural *executions* to process. This paper shows the mechanism of attaching plural *executions* to a process and describes the features of plural *execution* and reports the evaluation of speed control mechanism of program execution, guarantee of service processing time and influence on other processes.

1. はじめに

近年，計算機の価格は低下し，広く普及しており，計算機の性能の向上により，マルチメディア応用プログラムを実行することが可能となった．マルチメディア応用プログラムでは，実時間性を保証することが重要であり，このため，オペレーティングシステム（以降，OS と略す）で実時間性を保証する様々なスケジューリング方式の研究が行われている．実時間性を保証する研究では，ハードウェアの性能を最大限に利用し，実時間性を保証する．このため，処理終了の期限を保証できるように，細かくプログラムの実行を制御する．たとえば，実時間性を要求する複数の応用プログラムの処理終了期限を保証する研究[1] や，周期的なタスクの実時間性保証と非周期的なタスクの期限をできるだけ保証するスケジューリング方式の研究[2] が行われている．また，マルチプロセッサシステムで実時間性を必要とするタスクをスケジューリングするための研究[3] も行われている．

また，複数のスケジューリング方式を共存させる方式研究も行われている．これは，プログラムの処理の内容により，スケジューリングポリシを決定できるものである．たとえば，マルチメディア応用プログラムのためのスケジューラリングの研究としては，文献 4) がある．この研究では，スケジューラを階層化しており，異種のスケジューラを混在させている．また，複数のスケジューリング方式を 1 つのシステムに共存させる研究[5]，応用プログラムの動的な集合にスケジューリングの保証を提供する方式の研究[6]，および処理の進行の割合をモニタし，フィードバックすることでスケジューリングする方式の研究[7] も行われている．

我々は，プログラムへのプロセッサの割当てを制御することによるプログラムの実行速度を調整する研究を行っている[8),9)]．これは，プログラムがプロセッサを利用できる割合を保証するものである．プロセッサ

† 九州大学大学院システム情報科学研究科
Graduate School of Information Science and Electrical Engineering, Kyushu University

の性能とプログラムの処理時間が分かれば，プログラムの許容実行時間を保証することで，プログラムの処理終了期限を保証することができる．また，我々は，***Tender*** オペレーティングシステム[10] に，プロセッサの割当て単位である資源「演算」を実現し，プログラムの実行速度を調整する方式と優先度による方式の2つのスケジューリング方式を共存させた．資源「演算」の導入により，「演算」と関連付けられたプロセスのみが，スケジューリング対象となり，「演算」の持つプロセッサを利用できる程度により，スケジューリングされる．資源「演算」によるプログラムの実行速度調整機能を実現し，評価を行い，プログラムの実行速度調整機能の精度と割込みマスクにより受ける影響を明らかにした[11]．さらに，複数のスケジューリング方式を共存させることも可能にし，演算とプロセスを 1 対 1 で関連付ける基本的な評価を行った．先にあげた研究 4)~11) では，プロセスを対象としてスケジューリングしている．このため，1 つのプロセスに対し適用可能なスケジューリングポリシは，1 つであった．一方，演算は，プロセッサの割当て単位であり，スケジューリングポリシを保有する．1 つのプロセスに複数の資源「演算」を関連付けることを可能にすれば，1 つのプロセスに対して複数のスケジューリングポリシを適用できる．

本論文では，複数の資源「演算」を 1 つのプロセスに関連付ける機構を述べ，プロセスがプログラムの実行速度を調整する演算（以降，性能調整の演算と呼ぶ）と優先度を持つ演算（以降，優先度の演算と呼ぶ）を 1 つのプロセスに複数個関連付ける利点と特徴を述べる．性能調整の演算を複数の演算に分割して確保した場合のプログラム実行速度調整の精度と処理の均一性の評価，複数の優先度の演算を 1 つのプロセスに関連付けた場合のプログラムの処理時間，および性能調整の演算と優先度の演算を 1 つのプロセスに関連付けることによるサービス処理時間の保証と他プロセスへの影響の評価を報告する．

2. *Tender* における資源の分離と独立化

Tender では OS の操作する対象を資源として，分離し独立化した．各資源のプログラムの呼び出しインタフェースとなる表プログラム構造部は，ソフトウェア部品化されたプログラム群（以降，プログラム部品と名付ける）とそのプログラム部品をポインタで管理しているプログラムポインタ表からなる．プログラムポインタ表は，配列形式であり，各配列の要素は，プログラム部品へのポインタを持つ．プログラムへの制御の移行は，プログラムポインタ表を介して行われる．このようなプログラム構造により，各プログラム部品の動作状況の把握が可能になる．

資源には，資源名と資源識別子を付与し，資源操作のインタフェースを統一している．さらに，各資源を操作するプログラム部品を資源ごとに分離し，共有プログラムを排除した．また，各資源の管理情報も資源ごとに分離し，各資源の管理表の間のポインタを禁止した．

このように，資源の分離と独立化を行うことで，資源の事前生成や保留により，資源の作成や削除をともなう処理を高速化している．また，OS の動作や内部状態の理解や把握が容易になり，OS の理解を支援できる．さらに，プログラムを部品化できるため，機能の追加や変更が容易になっている．

3. 資源「演算」

3.1 資源「演算」の役割と機能

資源「演算」とは，プロセッサを利用でき，プログラムを実行できる程度（以降，演算の程度と名付ける）を持つ資源である．演算をプロセスと関連付けることにより，プロセスの実行が可能になる．つまり，利用者は，プロセスと演算を生成し，プロセスに演算を関連付けることで，演算が持つ演算の程度に応じてプロセスを走行させることができる．このため，演算の程度を変更することで，プロセッサを利用できる程度を変更でき，プロセスを生成し，演算の関連付けを行わないことによるプロセスの作り置きや，演算の関連付けと解除によるプロセスの実行の停止と再開が容易に実現できる．以降，プロセスに演算が関連付けられていることをプロセスが演算を持つと記述する．

演算管理のインタフェースには，生成，削除，プロセスへの関連付け，プロセスへの関連付け解除，走行プロセスの切替え，プロセスの状態を WAIT 状態に変更，プロセスの状態を READY 状態に変更，および演算程度の変更の 8 つがある．

3.2 演算の程度

演算の程度とは，プロセッサを利用できる程度を表すもので，各演算ごとに演算の程度を持っている．この演算の程度には，2 種類のものがある．1 つは，プログラムの実行速度を調整するものである．もう 1 つは，できる限り多くの演算程度を要求するもので，優先度を持つものである．

3.2.1 性能調整の演算

性能調整の演算は，プログラムの実行速度を調整する性能（1〜100%，プロセッサそのものの性能を 100%と

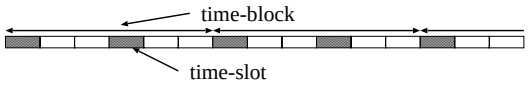


図 1 タイムブロックとタイムスロットの関係
Fig. 1 Relation between time-block and time-slot.

する，1%単位で指定）を持つ．性能調整の演算が持つ性能の総計が 100%以下の範囲で生成可能である．この総計が 100%を超えるような性能調整の演算の生成や性能の変更はできない．プログラムの実行速度の調整は，ある単位時間（これをタイムスロットと名付ける）で，プログラムの実行と停止を繰り返すことで実現できる．タイムスロットとタイムブロックの関係を**図 1** に示す．一定の連続したタイムスロットをタイムブロックと名付ける．タイムブロック中で，プログラムの実行速度を調整する要求性能に見合う時間分のタイムスロットを演算に割り当て，そのタイムスロット時間だけ演算を持つプロセスを走行させることで，プログラムの実行速度を調整できる．タイムスロットの長さは固定長であり，その長さは OS のコンパイル時に決定できる．

性能に見合う分のタイムスロットをタイムブロックの中から選択する方式として，これまでに我々は 3 つの方式を提案し，評価した[11]．これまでに提案した方式は，タイムスロットの割当て処理が高速であるが，処理の均一性に問題があった．そこで，処理の均一性の問題を解決した改良疑似周期割当て方式を考案し，採用する．処理の均一性とは，プログラムの実行速度を調整したときの処理の滑らかさを表すものである．改良疑似周期割当て方式は，要求性能に基づき n 個のタイムスロットを演算に割り当てる場合，i 番目に割り当てるタイムスロットの位置を，(総タイムスロット数) $*$ $(i-1)/n$ とする方式である．この方式では，一番最初に割り当てるタイムスロット位置が必ず 0 になる．このため，性能調整の演算を 2 個以上生成する場合に，必ず最初のタイムスロットの割当て位置の衝突が起こる．これを回避するために，タイムブロックの先頭から最初の空きタイムスロットを検索し，その位置を起点として，タイムスロットを割り当てる．これにより，最初に割り当てるタイムスロットの位置が衝突する問題を解決できる．

3.2.2 優先度の演算

優先度の演算は，生成されたときに，スケジューリングの優先順位となる優先度の値を持つ．優先度の演算には，2 つの状態がある．1 つは，プロセスと関連付けられていない状態で，優先度キューにはつなげない状態で存在する．もう 1 つは，優先度の演算がプロセスに関連付けられている状態で，優先度キューにつなげられていて，スケジューリング対象となる．

3.3 複数の演算生成

性能調整の演算を複数確保する場合の問題点として，次の 2 つの問題がある．1 つは，複数の性能調整の演算にタイムスロットを割り当てる際に，割り当てるタイムスロットの位置が衝突するという問題である．そこで，どの演算のタイムスロットの割当てを優先するのかを考える必要がある．タイムスロットの割当て位置の衝突により，本来の位置にタイムスロットが確保できないことにより，処理の均一性が低下することが考えられる．このため，タイムスロットの割当ての優先基準として，処理の均一性への影響を考慮しなければならない．そこで，確保する性能の大きさをタイムスロットの割当て優先基準として考え，性能の小さい演算のタイムスロットを優先して割り当てる．なぜならば，確保する性能が大きければ，割当てタイムスロット数が多く，タイムスロットが本来の割当て位置に割り当てられないタイムスロットがあったとしても，処理の均一性への影響は小さいと考えられるからである．

2 つめは，演算の生成と削除を繰り返し，タイムスロットの確保と解放を繰り返すうちにタイムスロットが散在し，処理の均一性が悪くなるという問題である．たとえば，タイムスロットの割当て位置の衝突により，一度本来の位置とは別のところに割り当てられると，他の演算が削除されても，タイムスロットの割当て位置はそのままとなる．これらの問題を解決するために，性能調整の演算を生成する場合や，性能調整の演算の性能を変更する場合は，いったんすべてのタイムスロットを解放し，性能の小さいものから優先してタイムスロットを割り当てる．こうすることで，処理の均一性の低下を防ぐことができる．

次に，性能調整の演算の生成時や性能の変更時のタイムスロットの再割当て時間について述べる．タイムスロットの再割当て処理は，タイムブロックの先頭からの空きタイムスロット検索処理とタイムスロットの割当て処理からなる．

空きタイムスロット検索処理を高速化するため，タイムスロットの再割当て処理では，先頭から最初の空きタイムスロット位置を保持する．その位置にタイムスロットを割り当てた場合には，その位置から後方の一番近い空きタイムスロットを検索し保持する．この効果を明らかにするため，空きタイムスロットの検索時間の評価を行った．PentiumII 450 MHz の計算機を使用し，***Tender*** 上でユーザプログラムを走行させ，演算の生成システムコールの処理時間を測定した．1

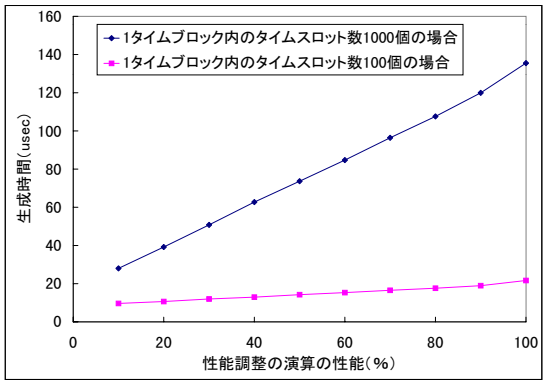


図 2 性能調整の演算の生成時間
Fig. 2 Creation time of "regulating" execution resource.

つのタイムブロック内のタイムスロット数が 1000 個の場合，100%の性能を持つ性能調整の演算の生成に改良疑似周期割当て方式をそのまま適用した場合，その時間は，135.5 usec であった．また，10%の性能を持つ演算が 9 個存在するときに，10%の性能を持つ演算を 1 つ生成する処理の時間は，138.1 usec であった．この両者の処理の違いは，空きタイムスロットを検索処理の有無である．空きタイムスロットの検索処理時間は約 2.6 usec であり，全体の処理時間の約 2%である．このため，空きタイムスロットの検索処理は，全体の再割当て時間に比べ小さいといえる．

次に，タイムスロットの割当て処理の評価を行う．上記と同じ環境において 1 つのタイムブロック内のタイムスロット数が 1000 個と 100 個のときの性能調整の演算の生成時間（タイムスロットの割当て処理を含む）を測定した．測定結果を**図 2** に示す．横軸は，生成する性能調整の演算の性能で，この性能の大きさにより割り当てるタイムスロットの数が変化する．図 2 より，改良疑似周期割当て方式の性能調整の演算の生成時間は，タイムスロットの割当て数に比例し，タイムスロット数が 1000 個のときの演算の生成時間は，28 usec（要求性能 10%）から 135.5 usec（要求性能 100%）である．

したがって，タイムスロットを再割当てする処理時間の大半は，タイムスロットの割当て処理時間である．このため，タイムスロットの再割当て処理時間は，再割当てする演算に割り当てられたタイムスロット数の総和に比例するといえる．再割当てするときの演算のタイムスロット数の総和は，最大でも 100%の性能を確保したときのタイムスロット数である．このことから，タイムスロットの再割当て時間は，最大でも図 2 と同じ程度の時間で処理ができる．

タイムスロットの割当て位置が衝突した場合には，衝突位置の後方の最も近い空きタイムスロットを検索し，そのタイムスロットを割り当てる．

3.4 スケジューリングの契機

スケジューラは，タイムスロットの境界でタイマ割込みにより呼び出される．スケジューラは，次のタイムスロットが性能調整の演算に割り当てられているかを調べる．もし，割り当てられていた場合は，その演算がプロセスに関連付けられているかを調べる．プロセスに関連付けられていた場合，関連付けられたプロセスの状態を調べ，READY または RUN 状態のときはそのプロセスを走行させる．それ以外の場合，つまり次のタイムスロットに割り当てられた性能調整の演算に関連付けられたプロセスが走行可能状態でない場合，優先度の演算の持つ優先度により走行させるプロセスを選択する．性能調整の演算を持つプロセスが，タイムスロットを使い切る前に WAIT 状態になった場合には，そのタイムスロットの残りの時間を利用するプロセスを優先度により選択する．

優先度の演算には，タイムスライス時間が設定されており，タイムスライス時間を使い切るとスケジューラが呼び出され，走行するプロセスが切り替えられる．優先度の演算のみを持つプロセスは，性能調整のプロセスが走行しているために，RUN と READY を繰り返す．このため，タイムスライス時間は，性能調整のプロセスの走行時間を除いた，実際にプロセッサを利用した時間のみカウントされる．また，プログラムの実行速度調整の精度を上げるために，タイムスライスの境界とタイムスロットの境界を合わせる必要がある．

3.5 特　　徴

資源「演算」の導入により，プロセスの生成の高速化，プロセスの処理途中での停止や再開，データ部を初期化することによる再起動の 3 つが可能となる[9]．

プロセスが複数の演算を持つことで，次の 3 つが可能となる．

(1) プロセスが複数の性能調整の演算を持つことができる．これにより，性能を分割して，複数の性能調整の演算を持つことで，複数のプロセス間で演算を使い回すことによるユーザレベルでのスケジューリングが可能となる．たとえば，あるプロセスグループで性能調整の演算を複数に分けて確保しておき，グループ内のプロセス間で，演算の割当てを変えることにより，スケジューリングが可能となる．

(2) 複数の優先度の演算を持つことで，優先度を上げることなくプロセスの走行時間を増やすこと

ができる．優先度の演算を生成し，複数の優先度の演算を持つことで影響を受けるプロセスは，同一優先度以下の優先度の演算を持つプロセスのみで，優先度の高い演算を持つプロセスに対しては影響を与えない．ただし，優先度の異なる演算を 1 つのプロセスに複数割り当てても，優先度の一番高い演算だけが有効になるので，異なる優先度の演算を割り当てることは意味がない．

(3) プロセスが性能調整の演算と優先度の演算を同時に持つことで，プログラムの最低の実行性能を保証できる．このとき，性能調整の演算で最低性能分を走行させ，優先度の演算で性能調整以外の時間を走行させる．

3.5.1 プロセスと演算の関係

演算とプロセスの関係を**図 3** に示す．1 つのプロセスに対し，複数の演算を関連付けることができる．逆に，1 つの演算を複数のプロセスに関連付けることはできない．各プロセスは，関連付けられた演算が持つ演算の程度の合計に従ってスケジューリングされる．

プロセスが複数の演算を持つ仕組みについて述べる．プロセスと演算の関連付けの対応関係を，資源「演算」の管理表に保持する．演算管理表は，各資源「演算」ごとに，演算識別子，演算の程度，関連付けられたプロセスの資源識別子などの情報を持ち，さらに，タイムスロットの割当て状態，優先度キューの状態と，各プロセスごとのコンテキスト情報を保持する．演算とプロセスの関連付けは，演算識別子とプロセス識別子の対応関係を演算管理表に持つことで実現する．

このため，プロセスと演算の関連付けと関連付けの解除は，簡単に行える．また，演算を他のプロセスに関連付け直すことも，簡単に行える．

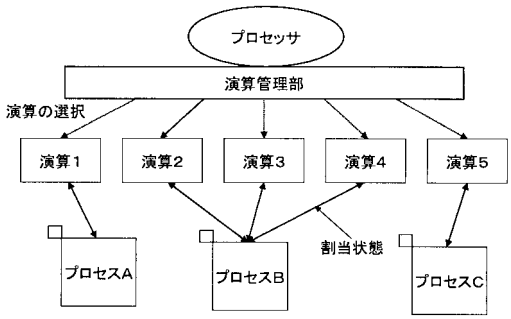


図 3 演算とプロセスの対応関係
Fig. 3 Relation between process resource and execution resource.

4. 評　　価

4.1 評価の環境と観点

プロセスが複数の演算を持つ場合の評価を行う．演算には，性能調整の演算と優先度の演算の 2 種類のものがあり，プロセスが演算を持つ 3 つの組合せと性能保証プロセスが他プロセスへ与える影響を評価した．

(1) 性能調整の演算を複数持つ場合
(2) 優先度のプロセスを複数持つ場合
(3) 性能調整と優先度の演算を同時に複数持つ場合
(4) 他プロセスへの影響

PentiumII 450 MHz の計算機を使用し，***Tender*** 上で測定用のユーザプログラムを走行させ，測定した．タイムブロックの長さは 1 秒で，1 タイムブロック内のタイムスロット数は 1000 個である．また，タイムスライス時間は 1 秒である．測定用のユーザプログラムは，コンソールに文字を出力するシステムコールを 100000 回繰り返すプログラムを使用した．また，速度調整度（説明は後述する）の測定には，コンソールへの文字の出力処理と変数のインクリメント処理を繰り返すプログラムを使用し，コンソールへの出力間隔数を 50 とし，速度調整度を算出し，その標準偏差を求めた．時間の計測には，プロセッサのクロック数をカウントするカウンタを用いた．

上記 (1) では，同じ性能を 1 つの演算で確保する場合と複数の演算で確保する場合の評価を行った．同じ性能を分割して確保する演算の数により，割当てタイムスロット位置が異なるため，性能を調整したときの処理時間と処理の均一性に影響があることが考えられる．そこで，性能調整をしたプログラムの処理時間と，処理の均一性を示す速度調整度を各 I/O の間で測定し，速度調整度の標準偏差を算出した．

上記 (2) では，プロセスが持つ優先度の演算の数による，処理時間の変化を測定した．

上記 (3) では，他プロセス数を増やし，プログラムの実行性能を保証することができるかどうかを評価する．また，他プロセス数により，プログラムの処理時間がどの程度変化するのかを測定し，性能調整の演算と優先度の演算を同時に持つことの効果について評価する．また，性能調整の演算と同時に持つ優先度の演算の数を変化させ，効果にどの程度の変化があるのかを評価した．また，他プロセスから受ける影響を，他プロセスの数を変化させ測定した．

上記 (4) では，性能保証プロセス存在するときの他プロセスの処理時間を測定し，性能保証プロセスが他プロセスに与える影響を測定し評価した．

速度調整度を，次のように定義する．プログラムの実行速度を調整しない場合の入出力時刻の間隔を t，要求性能 n%のときの入出力時刻の間隔を tn とするとき，速度調整度（η_i）を以下に示す．

$$\eta_i = tn_i/t_i/(n/100) = \frac{tn_i}{t_i} \times \frac{n}{100} \qquad (1)$$

速度調整度は，1 に近いほど処理を要求性能で調整していることを示し，1 を超えると要求性能以下，1 未満では要求性能を超える性能に調整していることを示す．つまり，η_i が 1 に近いほど，調整がうまくいっているといえる．また，速度調整度の標準偏差が小さいほど，処理の均一性は高い．

4.2 性能調整の演算を複数持つ場合

性能 30%を確保するために 30%性能を持つ演算，3 個の 10%性能を持つ演算，5 個の 6%性能を持つ演算，6 個の 5%性能を持つ演算，10 個の 3%の性能を持つ演算，および 30 個の 1%性能を持つ演算を確保した場合の処理時間を測定した結果を**図 4** に示す．理論値とは，性能 100%のときの処理時間に 100/30 を乗算して求めた値である．図 4 から，演算の個数に関係なく，理論値とほぼ同じ処理時間に調整できており，うまく処理速度を調整できていることが分かる．このことから，同じ性能を複数の演算に分割して確保しても，処理時間への影響はほとんどないことが分かる．

性能調整の演算を複数に分割して確保した場合の処理の均一性の評価を行う．各 I/O 間の速度調整度の標準偏差を求めた結果を**図 5** に示す．図 5 より，性能調整の演算を分割せずに 1 つの演算で確保した場合は速度調整度の標準偏差が小さく，処理の均一性が高いことが分かる．これは，各タイムスロットが連続することなく分散して割り当てられるためである．それに対し，いくつかに分割して性能調整の演算を確保した場合には，タイムスロットが連続してしまう．これは，先頭位置をずらしてタイムスロットを割り当てる方式のため，同じ性能の演算を複数確保すると，割当て開始位置が隣接してしまい，同じ間隔でタイムスロットを割り当てるため，タイムスロットが連続する部分が存在してしまう．このため，速度調整度の標準偏差が大きくなり，処理の均一性が低い．しかし，性能調整 30%のときの最悪の場合でも，標準偏差の値は約 0.015 であり，速度調整度のばらつきが小さく，複数の性能調整の演算で性能を確保しても，処理の均一性は高いといえる．

今回は，コンソールへの文字を出力を繰り返すプログラムで測定を行ったが，これまでに，プロセッサの処理時間と I/O の処理時間の割合を変化させ，プログラムの実行速度の調整機能の評価を行った[8]．その結果から，I/O の処理時間の割合が小さいほど，要求プロセッサ性能に近い時間に調整できることを明らかにした．本研究でのプログラムの実行速度の調整は同じメカニズムで行っており，同様の結果が出ると考えられる．

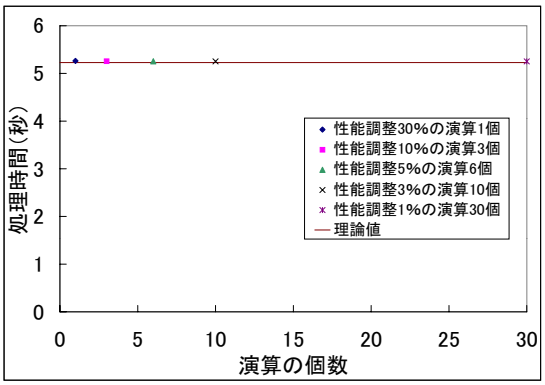


図 4 性能調整の演算を分割したときの処理時間
Fig. 4 Processing time on using multi “regulating” execution resources.

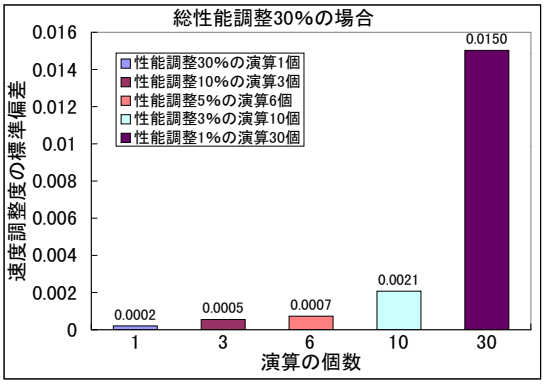


図 5 性能調整の演算を分割したときの処理の均一性
Fig. 5 Uniformity of processing on using multi “regulating” execution resources.

4.3 優先度の演算を複数持つ場合

1 つのプロセスが複数の優先度の演算を持つ場合の処理時間の評価を行う．優先度の演算の個数を変化させ，処理時間を測定したときの結果を**図 6** に示す．このときの優先度の演算と同一優先度の演算を 1 つ持つ他プロセスの個数は 3 つである．測定対象のプロセスよりも，優先度の高い演算は存在しない状態で測定した．

図 6 より，プロセスが持つ優先度の演算を増加させることにより，処理時間が短くなることが分かる．このことから，優先度の演算を持つ個数を変えることで，プロセッサの利用時間を変えることができるとい

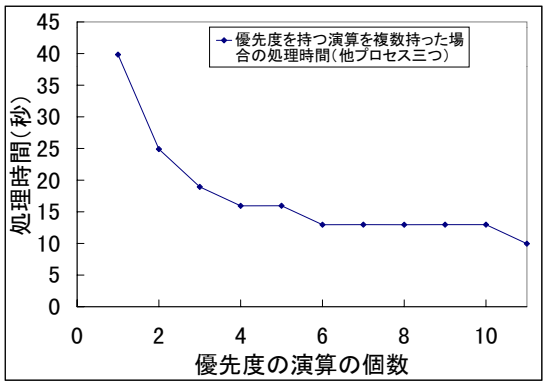


図 6 優先度の演算の個数と処理時間の関係
Fig. 6 Processing time on using multi "priority" execution resources.

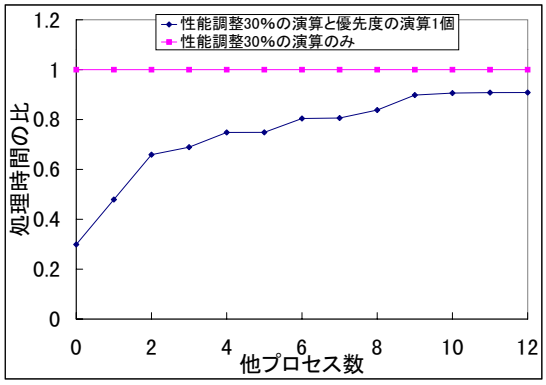


図 7 処理時間の保証の効果（性能調整 30%の場合）
Fig. 7 Effect of a guarantee of processing time ("Regulating" execution resource is 30%).

える．また，優先度の演算の個数が少ないときには，優先度の演算を増やすことによる処理時間の短縮効果が大きいが，優先度の演算の個数が増えるにつれ，グラフの傾きは緩やかになり，効果が小さくなることが分かる．

4.4 性能調整と優先度の演算を同時に複数持つ場合

4.4.1 処理時間の保証

測定対象のプロセスが，性能調整 30%の演算と優先度の演算をそれぞれ 1 つずつ持ち，測定対象のプロセスの優先度の演算と同一優先度の演算を 1 つ持つプロセスの個数を変化させたときの測定対象のプロセスの処理時間の測定結果を**図 7** に示す．このとき，測定対象のプロセスよりも，高い優先度を持つプロセスは存在しない．図 7 の縦軸は，性能調整 30%のときの処理時間を 1 としたときの処理時間の比である．図 7 より，同一優先度を持つプロセス数が増えるにつれて，処理時間の比が 1 に近付くことが分かる．これは，他プロセス数の増加により，優先度の演算による処理時間が減少しているためである．性能調整の演算による処理時間は，他プロセスの影響を受けず一定であり，他プロセスが増えても，処理時間の割合が 1 を超えることはなく，実行性能を保証できていることが分かる．

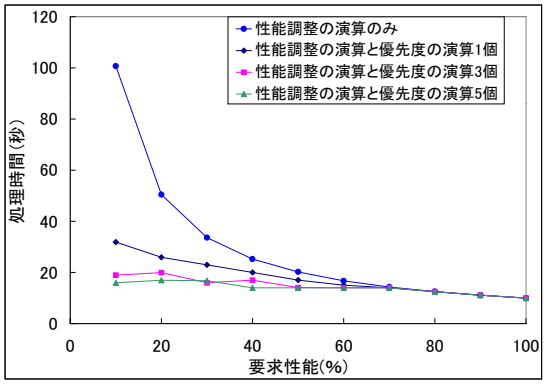


図 8 性能調整の演算と同時に割り当てる優先度の演算の個数と処理時間の関係
Fig. 8 Processing time on using one "regulating" execution resource and multi "priority" execution resources.

4.4.2 性能調整の演算と同時に持つ優先度の演算の個数の効果

性能調整の演算と優先度の演算を持つプロセスの性能調整の演算の性能を変化させ，処理時間を測定した結果を**図 8** に示す．パラメータとして，性能調整の演算と同時に持つ優先度の演算の個数を変化させた．このとき，測定対象のプロセスよりも，高い優先度を持つプロセスは存在しない．測定対象のプロセスと同一優先度の演算を 1 つ持つ他プロセスの数は 3 つである．図 8 より，性能調整の演算の程度が 70%以上のときは，性能調整の演算と同時に持つ優先度の演算の個数に関係なく，処理時間は等しくなることが分かる．これは，性能調整の演算により確保された時間と優先度の演算 1 つ分の時間で，処理が終了し，2 つめ以降の優先度の演算が使われず，優先度の演算の個数による差が出ないためである．性能調整の演算の性能が小さい場合には，処理時間に占める性能調整の時間の割合が小さくなり，優先度の演算による処理時間の割合が大きくなるため，優先度の演算の個数による処理時間の差が現れる．特に，性能調整の演算の要求性能が小さい場合には，優先度の演算を同時に持つ効果が大きいことが分かる．また，優先度の演算のみを複数持つ場合（4.3 節）と同様に，優先度の演算を増やしたときの処理時間の短縮効果は，同時に持つ優先度の演算の個数が少ない場合ほど大きい傾向がある．たとえば，要求性能 20%のとき，同時に持つ優先度の演算の個数が 3 個と 5 個のときの差は 3 秒であるが，1

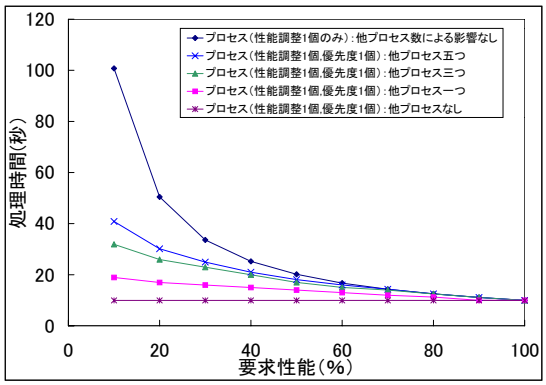


図 9 性能調整と優先度の演算を 1 つずつ持つプロセスの処理時間
Fig. 9 Processing time on using one "regulating" execution resource and one "priority" execution resource.

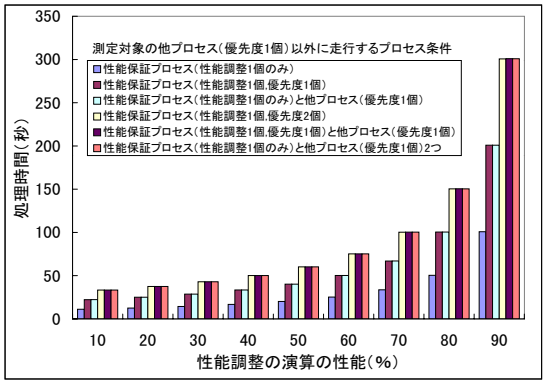


図 10 優先度の演算を 1 つ持つプロセスが他プロセスから受ける影響
Fig. 10 Influence of other process on a process using one "priority" execution resource.

個と 3 個のときの差は約 6 秒と効果が大きくなる．

4.4.3 他プロセスから受ける影響

性能調整の演算と優先度の演算を 1 つずつ持つプロセスが，同一優先度を持つ他プロセスのから受ける影響についての測定の結果を**図 9** に示す．パラメータとして，他プロセスの個数を変化させた．このとき，測定対象のプロセスよりも，高い優先度を持つプロセスは存在しない．図の実線は，性能調整の演算のみを持つプロセスの処理時間である．図 9 より，処理時間は他プロセスの個数が増加するほど，大きくなり，性能調整の演算のみを持つプロセスの処理時間に近付く．これは，優先度の演算により，処理をできる時間が少なくなるためである．また，性能調整の演算の性能が小さい場合には，優先度の演算を同時に持つ効果が大きい．性能調整の演算のみを持つプロセスの処理時間を超えておらず，前で述べたとおり，プログラムの性能を保証できている．

4.5 他プロセスへの影響

性能保証プロセスが走行することによる他プロセスへの影響を評価する．測定は，性能保証プロセスが 1 つと優先度の演算 1 個を持つ他プロセスが 1 つ以上存在する環境で行った．他プロセス中の 1 つのプロセスを測定対象のプロセスとした．性能保証プロセスは，性能調整の演算を 1 個と優先度の演算を 0 個以上持つ．性能保証プロセスと他プロセスが持つ優先度の演算の優先度は同じである．測定のパラメータとして，次の 3 つを変化させた．1 つは，性能保証プロセスが持つ性能調整の演算の性能である．2 つめは，性能保証プロセスが持つ優先度の演算の個数である．3 つめは，優先度の演算を 1 つ持つ他プロセスの個数である．これらのパラメータを変化させ，測定対象の他プロセスの処理時間を測定した．測定結果を**図 10** に示す．

図 10 より，性能調整の演算の性能（図の横軸）が大きくなると，他プロセスの処理時間は増加することが分かる．これは，性能調整されている時間以外の時間を優先度の演算で分割して利用するためである．性能調整の演算の性能 80%以上では，他プロセスへの影響が大きく，他プロセスの処理時間が大幅に増加する．

また，優先度の演算の個数による影響も大きく受ける．測定対象のプロセスの優先度の演算の個数は 1 つであるが，他のプロセスの優先度の演算の個数の合計の個数により，処理時間が変化する．これは，すべてのプロセスに割り当てられている優先度の演算の合計の個数により，1 つの優先度の演算あたりの処理時間が決められるからである．つまり，すべてのプロセスに割り当てられた優先度の演算により，性能調整以外の時間を分割している．このことから，一番高い優先度の演算を持つプロセスの平均実行時間は次の式で表される．

$$
\begin{aligned}
&(\text{プロセスの平均実行時間}) \\
&\quad = (\text{性能 100\%のときの処理時間}) \\
&\qquad \times \frac{100}{(100 - \sum(\text{性能調整の演算の性能 (\%)}))} \\
&\qquad \times \frac{\sum(\text{対象プロセスと同一優先度の演算の個数})}{(\text{対象プロセスの優先度の演算の個数})}
\end{aligned}
$$

5. ま と め

プロセッサの割当て単位である資源「演算」の役割と特徴について述べ，1 つのプロセスに複数の演算を関連付ける機構と特徴について述べた．性能調整の演算と優先度の演算の 2 種類の演算を複数組み合わせて利用する方式について述べ，評価を行った．

複数の性能調整の演算を 1 つのプロセスに関連付け

る場合，1 つの演算で実行速度を調整するときと複数の演算で実行速度を調整するときでは，調整した処理時間にほとんど差がなく，うまく実行速度を調整できることを示した．また，処理の均一性の面では，1 つの演算で性能を確保する場合が，タイムスロットが分散して確保され，処理の均一性が一番高い．しかし，性能 30%を複数の演算で確保したのときの最悪の場合（性能 1%の演算が 30 個）でも，速度調整度の標準偏差は 0.015 と十分小さく，処理の均一性は十分高いといえる．

複数の優先度の演算を 1 つのプロセスに関連付ける場合は，そのプロセスよりも優先度の高いプロセスに影響を与えずに，プロセッサの利用時間を増加させることができることを示した．その効果は，優先度の演算の個数が少ないプロセスに新たに優先度の演算を関連付ける効果が大きく，ある程度の優先度の演算を関連付けられたプロセスには，新たに優先度の演算を関連付ける効果が小さい．

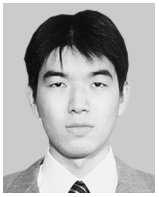
性能調整の演算と優先度の演算を 1 つの演算に関連付ける場合は，1 つのプロセスに性能調整の演算と優先度の演算を同時に関連付けることで，プログラムの実行性能を保証できることを示した．性能調整の演算の性能が小さい場合，優先度の演算を同時に関連付けることで，プロセッサの空き時間を利用する効果が大きいこと示した．性能保証プロセス以外のプロセスの個数により，性能保証プロセスの受ける影響を明らかにした．また，性能保証プロセスが他プロセスに与える影響を評価し，性能保証の値が大きな場合には特に他のプロセスへ与える影響が大きく，OS 内の優先度の演算の個数の合計により，優先度の演算のプロセッサの利用時間が決定することを示した．

残された課題として，複数の演算で性能を確保した場合，プログラムのプロセッサと I/O の処理割合が処理時間へ与える影響の評価がある．

参 考 文 献

1) Jones, M.B., Rosu, D. and Rosu, M.: CPU Reservations and Time Constraint: Efficient, Predictable Scheduling of Independent Activities, *SOSP '97*: *16th ACM Symposium on Operating Systems Principles*, pp.198–211 (1997).
2) Shin, K.G. and Chang, Y.-C.: A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks, *IEEE Trans. Comput.*, Vol.44, No.12, pp.1405–1419 (1995).
3) Burchard, A., Liebeherr, J., Oh, Y. and Son, S.H.: New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems, *IEEE Trans. Comput.*, Vol.44, No.12, pp.1429–1442 (1995).
4) Goyal, P., Guo, X. and Vin, H.M.: A Hierarchical CPU Scheduler for Multimedia Operating Systems, *OSDI '96: USENIX Association 2nd Symposium*, pp.107–121 (1996).
5) Ford, B. and Susarla, S.: CPU Inheritance Scheduling, *OSDI '96*: *USENIX Association 2nd Symposium*, pp.91–105 (1996).
6) Baker-Harvey, M.: ETI Resource Distributor: Guaranteed Resource Allocation and Scheduling in Multimedia Systems, *OSDI '99*: *USENIX Association 3rd Symposium*, pp.131–144 (1999).
7) Steere, D.C., Goel, A., Gruenberg, J., McNamee, D., Pu, C. and Walpole, J.: A Feedback-driven Proportion Allocator for Real-Rate Scheduling, *OSDI '99*: *USENIX Association 3rd Symposium*, pp.145–158 (1999).
8) 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，Vol.J81-D-I, No.4, pp.386–392 (1998).
9) 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，Vol.J83-D-I, No.1, pp.184–193 (2000).
10) 谷口秀夫：分散指向永続オペレーティングシステム *Tender*，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.95, No.7, pp.47–54 (1995).
11) 田端利宏，谷口秀夫：***Tender*** オペレーティングシステムの資源「演算」によるプログラム実行速度調整機能の実現と評価，情報処理学会論文誌，Vol.40, No.6, pp.2523–2533 (1999).

(平成 11 年 12 月 13 日受付)
(平成 12 年 4 月 6 日採録)



田端 利宏（学生会員）

平成 10 年九州大学工学部情報工学科卒業．平成 12 年，同大大学院システム情報科学研究科修士課程修了．現在，同大大学院システム情報科学研究科博士後期課程在学中．オペレーティングシステムに興味を持つ．

谷口　秀夫（正会員）

昭和 53 年九州大学工学部電子工学科卒業．昭和 55 年同大大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和 62 年同所主任研究員．昭和 63 年 NTT データ通信（株）開発本部移籍．平成 4 年同本部主幹技師．平成 5 年九州大学工学部助教授．平成 8 年九州大学大学院システム情報科学研究科助教授．博士（工学）．オペレーティングシステム，分散処理に興味を持つ．著書「オペレーティングシステム」（昭晃堂）．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．