3P-9

TOP－1スケジューラに関する考察

大澤　暁　　穂積　元一
日本アイ・ビー・エム株式会社　東京基礎研究所

1. はじめに

TOP－1は小規模並列処理ワークステーションと位置づけられており、スレッド機構を用いて多数のプロセス、スレッドを同時に実行することができる[1,2]。本稿では、このような環境下における効率的かつシンプルなスケジューリング法について述べる。

2. プロセス・スケジューリング

汎用計算機では、オペレーティング・システム(OS)内部で何らかの形のプロセス・スケジューリングを行うのが一般的である。**TOP－1**も汎用ワークステーションであるから、OSにおいてスケジューリングを行う必要がある。

プロセスのスケジューリングにはいくつかの方法があるが、そのプロセスのプライオリティ、CPU実行時間等をもとに、動的に実行順序を調整するのが普通である。UNIX*でもこの方法がとられており、システム内に存在するすべてのプロセスのプライオリティを1秒に1回、次の3つの値をもとに再計算している。

1)　ユーザが指定するプライオリティ
2)　システムのロードアベレージ
3)　プロセスのCPU実行時間

特に4.3BSDでは、プライオリティ別に32本の実行待ち行列があり、優先度の高い待ち行列からラウンドロビン方式でCPUへの割当てが行なわれている。再計算の結果プライオリティが変わった場合には、新しいプライオリティに対応する待ち行列への付け替えが行なわれる。通常の単一CPUワークステーションにおいては、カーネルプロセス、ユーザプロセスのプリエンプションの違い、待ち行列の本数や再計算のアルゴリズムに多少の違いはあるものの、基本的には上述した方法でスケジューリングが行なわれている。

TOP－1のようなマルチプロセッサ・システムでは、システム内に多数のプロセス、スレッドが存在する確率が高く、全てのプロセス、スレッドをこのような方法を用いてプライオリティの再計算を毎秒行うと、プロセス、スレッドの数に比例して時間がかかり、極めて効率の低下することが予想される。

3. マルチプロセッサOS

一般に、汎用マルチプロセッサ・システムのOSは、それが実行される形態等によりいくつかに分類することができる。現在の**TOP－1**では、プロセス、スレッドのスケジューリングを含むカーネル・プロセスを単一の固定CPUが担当し、残りのCPUがユーザ・プロセスを実行する方式をとっている[2]。しかしこの方式では、各CPUにおけるディスパッチングや、プロセス、スレッドの管理をひとつのCPUで行うことになり、効率面で必ずしも好ましいとはいえない。そこで本稿では**TOP－1**およびその上で実行されるOSを想定し、プロセス、スレッドの管理を各CPUにおいて個別に行う方法について考察する。従って各CPUは、個別に専用の待ち行列を持つ。カーネル・プロセスで行うべき仕事には、システム・コールをはじめ、この他にもいくつかあるが、ここではディスパッチングを含めたプロセス、スレッドのスケジューリングに限定する。

4. プロセスの優先度とスレッドの優先度

TOP－1 OSでは、ユーザはひとつのプロセス内に複数のスレッドを作ることができる[3]。CPUでの実行単位はスレッドであるから、今回は実行可能なスレッドを待ち行列に登録する方法をとった。プライオリティの計算には次の項目を考慮する必要がある。

1)　プロセスのプライオリティとスレッドのプライオリティ
2)　各プロセス内のスレッドの数

最初に1)について検討する。スレッドはアドレス・スペースを共有するなど、プロセスに比べてその機能を簡素化したものと考えることができる。従って同一プロセス内のスレッド間で、プライオリティをフレキシブルに変える必要性はそれほどない。そこで今回は、スレッドのプライオリティは、それを含むプロセスのプライオリティで代表し、同一プロセス内のすべてのスレッドは同じプライオリティとして扱うこととした。

次に2)であるが、カーネルがスレッド単位でCPUへの割当てを行うと、同じプロセスでもスレッドをたくさん作ったプロセスのほうが、CPUに割当てられる機会が増えることになる。そこでプロセス内のスレッド数に応じてプライオリティを調整し、プロセス間で公平化を図ることにした。

5. イベント型スケジューリング

ワークステーションはホスト計算機とは異なり少人数で使用するので、プロセス間で大きな偏りがなければ厳格にプライオリティのコントロールをする必要はない。プライオリティの初期の目的に戻って考えてみると、CPUに割当てられて実行したらプライオリティを下げてより下位の待ち行列に登録し、ある時間たってもCPUに割当てられなかった場合にはプライオリティを上げて、割当てられる可能性を高くすれば良い。

TOP-1 Scheduling Method
Gyo OHSAWA and Motokazu HOZUMI
IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.

ここで取り上げるイベント型スケジューリング法は、この考え方に基づくものである。

1) プロセスが作成された場合、初期プライオリティを計算し、当該プライオリティに対応する各CPUの待ち行列のうち、最も短いところに登録する。
2) スレッドがCPUに割当てられた時には、プライオリティを一定値だけ下げる。
3) スレッドがsleepした時はwakeupする時に、sleepしていた時間に応じてプライオリティを上げる。
4) ある一定時間、待ち行列に登録されながら実行されなかったスレッドは、一定値プライオリティを上げる(図1)。プライオリティを上げるための時間間隔はシステムのロード・アベレージにより動的に変更する。
5) プロセス、スレッドが途中でスレッドを作成・消滅した時は、数の増減に応じてある程度プライオリティの調整を行う。

以上の方法を用いると、簡単かつ効率的なスケジューリングを行うことができる。1)の方法ではユーザが指定するプライオリティ(UNIXにおけるnice)を初期プライオリティに反映できる他、ロード・バランシングの効果もある。

2)でプライオリティを下げるのは、そのスレッドがIO等でCPUを放棄した時か、プリエンプションが起こった時になる。3)4)は一定時間CPUを使わなかった場合にプライオリティを上げるもので、特に4)は待ち行列チェインのヘッドのみのつなぎかえで済むので、待ち行列に登録されているスレッドの数に左右されず、短時間処理が可能である。

この方法を各CPUが独自に行うと、待ち行列に登録されてしまったスレッドの動的なロード・バランシングを行うことができず、今後の課題となる。ひとつの解決法としては、4)の時間間隔より少し長めの間隔で、ある特定のCPUがすべてのCPUの待ち行列を調整してロード・バランシングを行うことが考えられる。

6. 実OSでの実験

上述したスケジューリング法を実際にＡＩＸ ＰＳ／２*とTOP－１ OSに実装して簡単な評価を取った。しかし前者はシングルプロセッサ・システムであり、後者もスケジューリングは特定の単一CPUで行っているので、今回の実験ではロード・バランシング等、マルチプロセッサに起因する項目は除外した。また、スレッドは用いなかった。

実験では、ディスクIOや関数コール等いろいろなタイプのプログラムを多数、数時間にわたり実行した。また比較のために、毎秒1回プライオリティ再計算を行う従来の方法(Ordinal)も実行した。図2は、1回のプライオリティ再計算に要した時間(Ordinal)と、待ち行列のチェインつなぎかえに要した時間(Event)をまとめたものである。プライオリティ変更の回数を考慮するために、プライオリティ制御のルーティンが5秒のタイム・スパンの間に、どれくらいの時間実行していたかを調べたのが図3である。イベント型の有効性が確認できる。

以上はＡＩＸ ＰＳ／２で行ったものであるが、同種のプログラムをTOP－１上で実行するのに要した時間をグラフにすると図4のようになる。イベント型の方が20分近く短くなっていることがわかる。

7. おわりに

本稿では、TOP－１を対象とした高効率なスケジューリング法について述べ、簡単な評価結果を報告した。現在のTOP－１ OSには用いられていないが、さらに検討を加えている最中である。

[参考文献]
[1] 穂積:「TOP－１オペレーティング・システム -基本方針-」, 第39回情処全大論文集(1989)
[2] 山崎, 他:「同上 -プロセス・スケジューリング-」, 第39回情処全大論文集(1989)
[3] 吉永, 他:「同上 -スレッド機構-」, 第39回情処全大論文集(1989)

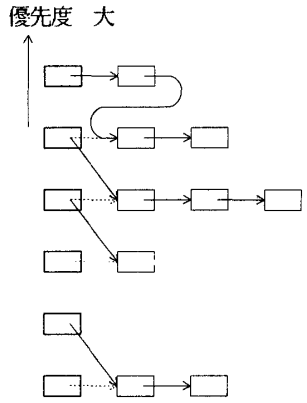

図1 待ち行列のつなぎかえ

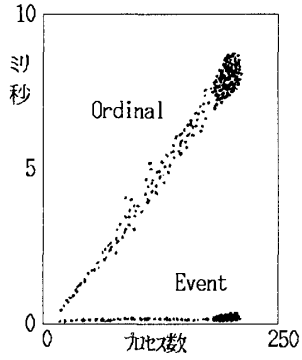

図2 優先度制御

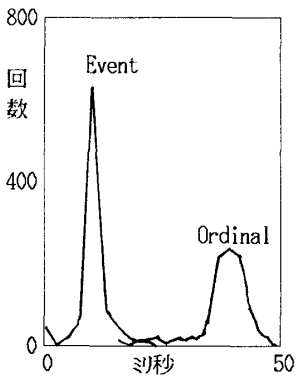

図3 優先度制御時間

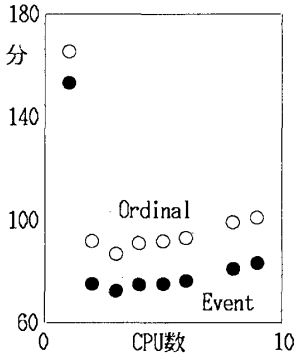

図4 プログラム実行時間(TOP-1)

*UNIXはAT&Tベル研究所の登録商標、AIX PS/2はIBM社の商標です。