記　号　処　理 73－6
（１９９４．３．１８）

Lisp Server におけるシステムコールキャッシュ

岩井輝男　中西正和
慶應義塾大学理工学研究科
計算機科学専攻中西研究室
横浜市港北区日吉 3-14-1

Lisp Server は Mach OS 3.0 と似た、単純な機能と Lisp インタプリタを備えた OS である。この基本となる考えはシンプルで、機能毎に thread に分割している。

Lisp インタプリタとシステムコールを処理する thread の通信について実行速度を測定して、通信に大きな時間がかかっていることが得られた。これをできるだけ最小にし、最適にするためにシステムコールの caching を行なうことによって実行速度を測定した結果、改善することが可能である。

System call caching of Lisp Server

Teruo IWAI and Masakazu NAKANISHI
Department of Computer Science
Graduate School of Science and Technology
Keio University
3-14-1, Hiyosi, Kouhoku, Yokohama 223, Japan

We propose caching of Lisp Server(LS) which is similar to Mach OS 3.0. LS is a kind of operating systems which have functions of simple Lisp interpreter. The basis idea is that it is simple, and distributes each functions to thread.

We measure performance of communications between threads, and find that these cost a lot. Caching of communictions makes times of communitions reduced more. We implement caching of system call, and it makes LS higher performance.

1 はじめに

Operating System の UNIX は最初はシンプルなものであったが、だんだん大きく、複雑な機能を付加してさらに複雑になってきている。例えば仮想記憶、ネットワーク通信、リモートファイルシステムの機能の付加を行なってきた。これにより、使い易さは向上しているが、UNIX のカーネルの変更、パラメータなどの変更が難しくなってきた。Micro kernel はこの複雑な UNIX カーネルを小さくすることを提案している。Unix の kernel のたくさんの機能をユーザプロセス (スレッド) に分割することで小さく、単純なものにすることが可能である。

2 Lisp Server の思想

Lisp Server[1][3] は Micro Kernel Mach OS[2] の理念の単純性、拡張性を採り入れ、ユーザが容易にカーネルの機能の変更可能なように設計されたものである。Lisp Server はマルチユーザ、マルチスレッドであり、Lisp の環境をユーザに与える。これにより、ユーザレベルで容易に Lisp のプログラムでカーネルの機能の変更、追加を行なうことが可能である。

3 Lisp Server の構成

Lisp Server は以下の機能がある。

- カーネル機能のスレッド化
- マルチユーザ マルチスレッド
- セル領域の共有

3.1 スレッド化

Lisp Server の機能をスレッドに分散させている。

- init のスレッド
- システムコールのエミュレートスレッド
- スレッドのスケジューリングスレッド
- ガーベージコレクション (GC) スレッド
- getty スレッド
- ネットワークスレッド

これらのスレッドはユーザのスレッドと同じレベルで実行する。すべてのスレッドをスケジューラースレッドがスケジュールしている。以下に各スレッドの役割を説明する。

3.1.1 システムコールスレッド

この Lisp Server はシステムコールがあり、これらのシステムコールをシステムコールスレッドが処理する。またシステムコールスレッドのみの処理ではできない時は Lisp Kernel に処理要求する。他のスレッドはシステムコールを発行した後にこのスレッドにスレッド switching を行い、システムコール要求を出したスレッドが実行状態になるまでこのスレッドは一時停止する。

システムコールは以下のように分類できる。

- I/O
  デバイス, ネットワークの I/O をサポートしている。

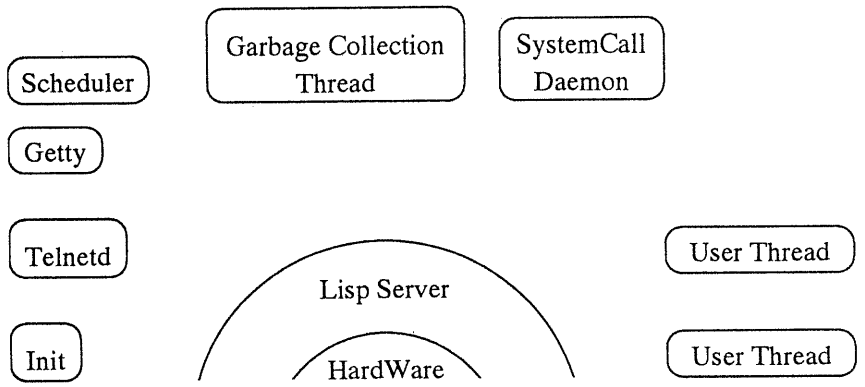

図 1: Lisp Server のシステムスレッド

- GC
  GC のプログラムを Lisp で記述するためのプリミティブを用意している。
- 時間
  プログラムの実行時間を計測するために現在の時間、および、1 つのスレッドの実行時間を取り出すための関数。
- スレッドの管理
  新規のスレッドの生成、破壊、停止、再起などを行なう関数。
- スレッド間の排他制御
  スレッド間で共有する変数の排他制御をおこなうためのセマフォ関数。
- LISP のセル、アトムの取りだし
  セル、アトムをフリーリストから取り出すときに呼び出す関数。既にとりだしたセルのアクセスはシステムコールを通さないで直接アクセスすることが可能である。

このようなシステムコールをシステムコールスレッドがサポートしている。これらのシステムコールはすべて Lisp の関数としてアクセス可能である。ただし、ハードウェアの I/O などは UNIX のシステムコールに変換している。またファイルシステムについても同様、UNIX のファイルシステムに変換を行ないエミュレートしている。

3.1.2 スケジューラスレッド

スケジューラはスレッドをラウンドロビンでスケジュールしている。このスレッドは C 言語で記述しているが、Lisp 言語で記述可能になるようにシステムコールでスレッドのアクセス関数をユーザに提供している。

3.1.3 GC スレッド

この GC は現在は C 言語で記述しているが、Lisp でも記述可能なようにシステムコールで GC を Lisp で記述可能になるようにしている。

3.1.4 init スレッド

このスレッドは他のすべてのスレッドの親スレッドである。このスレッドがシステムのスレッドを生成する。

3.1.5 getty スレッド

このスレッドはユーザにLispのトップレベルのread,eval,printのループをユーザに与える。

3.1.6 ネットワークスレッド

このスレッドはリモートホストからのネットワークでの接続を可能にするためにネットワークアクセスを監視している。またネットワークでは他のホストからはtelnetコマンドでLisp Serverに接続可能であり、Multi-User ,Multi-threadの環境を提供している。

4 インプリメント

Lisp Serverを現在SunOS4.1.2上にインプリメントしている。スレッドの実現方法はSunOSのLight Weigth Process(LWP)のライブラリを使用している。

デバイスドライバの部分はSunOSのシステムコールを呼び出すことでエミュレートしている。ただし、単にUNIXのシステムコールを呼び出すのではインプリメントできない。これにより、デバイスや、ネットワークのイベントの取りかたはpolling方式ではなく、割り込み方式で行なっている。これはLWPライブラリにある、agent機能を使ってインプリメントしている。このデバイスドライバのためにCPUが余計な消費することがないようにしている。以下の機能をもったライブラリがある。

5 システムコール caching

システムコールスレッドは他のスレッドからシステムコールの要求を受けつける。このスレッド間の通信はメッセージ通信を使用している。スレッドはシステムコールスレッドに対してメッセージでシステムコール要求を送って、その結果をシステムコールスレッドからメッセージで受けとる。本処理系で行っているシステムコールcachingはセルの取り出しである。他のシステムコールでcachingを行わない理由は以下のことが挙げられる。

- 要求回数がセルの取り出しに比べて非常に少ない。
- cachingが困難である。

consを行なう毎にメッセージ通信を行なってスレッドswitchingを行ない、システムコールスレッドが処理を行なってその処理を終了しなければそのスレッドは実行が再開されない。このように頻度の高いconsなどの処理毎にswitchingを起こしていることになる。本研究のLispではdeep bindingとなっている。このため高い頻度でセルの取りだしを行なうのでこのコストが大きいと実行速度に大きく影響する。

5.1 UNIX のメモリ管理

Lisp Serverではメモリの単位はセル(アトムなども含む)であるが、UNIXではpageである。よって、UNIXのpageサイズはLispのセルのサイズと比べると比較的大きい。しかもUNIX

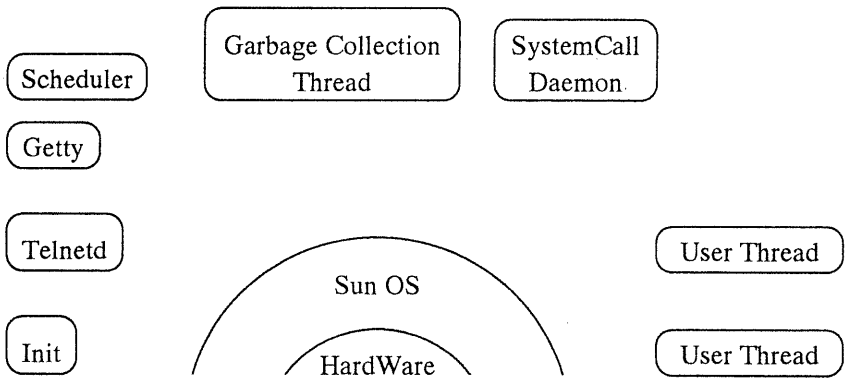

図 2: SunOS 上でのインプリメント

では demand paging を行なっているので必要になった時に始めてメモリ領域を割り当てる。

5.2 Lisp Server のメモリ管理

LispServer ではセルをとり出した後で必ず 1 度はセルに書き込むために demand paging はインプリメントする意味がない。また Lisp のセルのサイズはこの Lisp の場合は 12 バイトであり、UNIX の page サイズ (4096,8192 バイト) よりもかなり小さい。よって同じように Lisp Server と UNIX のシステムコールで同じ大きさのメモリの確保をおこなうことを比較した場合はシステムコールの回数が LispServer の場合が数百倍以上となる。よってシステムコールを行なったことによる、オーバーヘッドが大きくなる。

6 実験

6.1 caching の有効性

セルの取り出しの caching 行わない場合と行う場合のの実行速度を計測を行なった。使用したプログラムは Ackermann 関数と Quick Sort のプログラムである。実験環境は

表 1: caching の有り無しの実行時間の比較

プログラム	caching 無し (sec)	caching あり (sec)
Ackermann	5.512369	1.414618
QuickSort	3.524985	0.496792

- SPARC Station10 31(1CPU) Memory32Mbytes
- SunOS4.1.3

である。Ackermann 関数は (ack 3 3) を実行した。Quick Sort のプログラムは数値データ 72 個で Quick Sort アルゴリズムを用いたものである。

表 1からシステムコール caching を行うことで 6 倍程度速度が早くなることが判る。

6.2 cache サイズ

システムコールの caching の有効性が実験から得たことから最適な cache サイズを実験によって現在の Lisp Server の cache のサイズを変化させたときの実行時間について実験を行った。

図 3,4は以下の実行時間を表す。

Usertime	ユーザ時間:プログラムのスレッドだけの実行した時間
Systemtime	システム時間:Lisp Server のシステム時間
Realtime	実時間:実際にかかった時間

ユーザ時間とシステム時間の和は実時間になる。また、セルの量で実行時間が変化するので本実験では GC が起きない程度の量のセル使っている。また、システム時間とユーザ時間の和は実時間となる。

7 結果

Lisp Server でシステムコールの通信の負荷を減らすためにセルやアトムなどをとり出す時のシステムコールを caching することにより、通信回数を減らすことが可能である。この cache システムは 1 つのスレッドに 1 つの cache 領域を割り当てる。よって cache に入っているセルやアトムは他のスレッドから使用することができない。また、cache サイズを大きく割り当て過ぎると GC の回数が頻繁になって、更に遅くなる。また、thread 上のアプリケーションの種類によってもセルの消費量が異なり、多く消費するもの、ほとんど消費しないものがある。これらのスレッド全て同じ cache の量を割り当てるのは無駄が大きい。例えば実行してるスレッドの数が少ないときはそれほど無駄ではないが、ほとんどセルを使わないスレッドの数が大きくなった場合はセルが無駄になる。実験データから cache size は 100 以上になると実行時間はほとんど変わらないことが分かる。つまりシステム時間がほとんど 0 に近くなる。

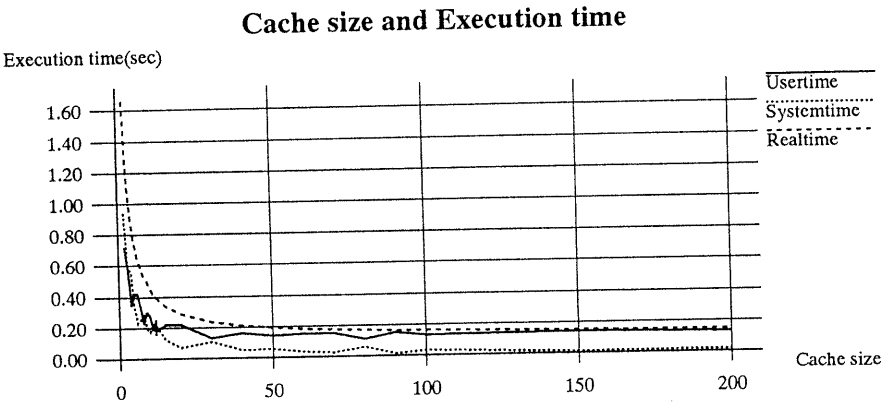

図 3: QuickSort での実行時間

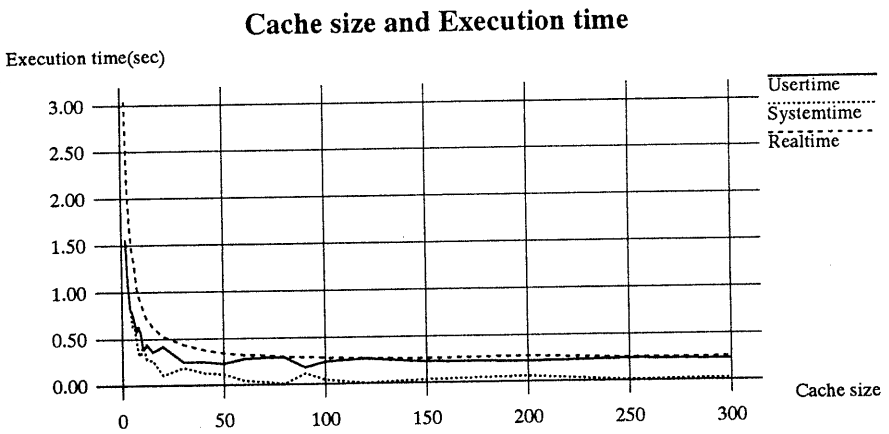

図 4: Ackermann 関数での実行時間

8 展望

現在のLispServerではcacheサイズを静的に固定であるがこれをスレッドのセルの取り出しの頻度に応じてcacheのサイズを動的に変化させる事で多くの無駄なセルをcacheに残したままGCを行うことができるだけないようにすることが可能である。本処理系ではdeep bindingを採用しているためにほとんどのプログラムはかなりのセルのとりだしを行なうようになっているが、deep bindingからshallow bindingへの変更、またはcompilerなどを使うことによって、program間のセルの取り出す頻度が変わると考えられる。

動的cache sizeの方法としては単位時間あたりのシステムコールの回数を常に計算して、その数値によってcacheの量を再計算するようにする。これにより、ほとんどセルを使用しないスレッドではcacheの量が少なく、多く使用するスレッドではcacheの量が多くなる。

今回のtime sliceの間隔は100msecと固定であるが、このtime sliceの最適な値と変化させたときの実行速度の関係についても研究を行なう。

参考文献

[1] Teruo IWAI and Masakazu NAKANISHI. Lisp server having an idea of micro kernel philosophy. *Proceedings of the Second International Conference PaCT93*, August 1993.

[2] Richard Rashid, Daniel Julin, Douglas Orr, Richard Sanzi, Robert Baron, Alessandro Forin, David Glouib, and Michael Jones. Mach: A system software kernel. *Proceedings of the 34th Computer Society International Conference COMPCON 89*, February 1989.

[3] 岩井輝男中西正和. Micro kernelの思想をとりいれたlisp serverの設計. 情報処理学会 記号処理研究会, June 1993.