記 号 処 理 69－2
（1993．6．18）

Micro Kernelの思想を採り入れたLisp Serverの設計

岩井輝男　中西正和
慶應義塾大学

Lisp Serverは Mach OS 3.0と似ていて、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、拡張性のあるカーネルの通信を持っている。この目的はすべてのOSの処理がカーネルを通してユーザプログラム(スレッド)間の通信で可能になるまで、OSの機能をカーネルの外に出すことである。極端に言えば、カーネルはユーザプログラム間の通信をサポートするだけとなる。

このような思想のもとに設計したLisp Serverは Multi-User Multi-Threadであり、すべてのLispのセルを共有した、環境をユーザに与えるOSである。

Design of Lisp Server having an idea of Mircro kernel philosophy

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 a **Lisp Server** which is similar to Mach OS 3.0. The **Lisp Server** is an operating system with the simple function and Lisp interpreter. The fundamental idea is that of a simple and extensible communication kernel. Its goal is to move more and more functionality out of the kernel, until everything is done by user program (thread) communicating via the kernel. In the extreme, the kernel must only provide other support besides user program to user program communication. **Lisp Server** is an operating system that gives environment of multi-user multi-thread and common memory of Lisp.

1 はじめに

Operating System の UNIX[1] 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 (thread) 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 分散 OS、Mach

Mach Operating System[2]-[1] は分散環境構築のために Carnegie Mellon University で設計されたもので、1 の CPU、または複数の CPU に対応するように作られている。拡張性と互換性を高めるために以下のような特徴がある。

- 新旧の CPU アーキテクチャに対応
  マルチプロセッサに対応していて、複数のジョブを複数のプロセッサに自動的に分散させ、1 つのプロセッサの計算機に比べて効率的にジョブを処理することが可能である。

- 新旧のメモリマネージメントアーキテクチャに対応
  今までは少しのメモリをできるだけ効率的に使うことが目的であったが、現在はハードウェアの進歩により、大容量のメモリが簡単に手にはいるようになり、Mach のメモリ管理は大容量のメモリを効率的に使うことに焦点が当てられている。

- 新旧の I/O とネットワークに対応

- 新旧の OS の環境に対応
  いろいろな OS との互換性をできるだけ保つようになっている。

どのようなシステムに対しても Mach のインプリメント可能なように設計されている。

OS の巨大化によって、CPU の変更、OS の機能追加、CPU,I/O などのハードウェアの変更のため OS の変更が高価なものになってきている。そのため OS の変更ができるだけ少なくて済むように設計されている。このような目的で作成された Micro Kernel Mach OS の基本理念は以下のような性質を持つことである。

- 単純性
  Mach OS では従来カーネルが処理する機能を分割してカーネルの外部で実現している。よってカーネルはできるだけ最小限度の機能しかもたない。

- 拡張性
  従来のカーネルの機能を外部で実現するようにしているため、ハードウェアの違いなどは最小限度の機能しかもたないカーネル部分のみの変更で済むようになる。

Micro Kenel Mach OS は複数の OS をサポート可能なように、OS 環境をユーザレベルのタスクで実現している。これをその OS の Server と呼び現在は 4.3BSD,POE,MS-DOS などがイ

[1] UNIX is a trademark of ATT Lab.

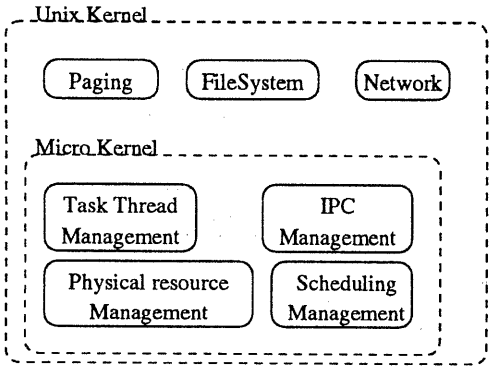

Figure 1: Mach OS の機能

ンプリメントされている。

Micro Kernel のは以下の機能を持っている。

- タスク、スレッドの管理
- タスク、スレッドのスケジューリング
- 仮想記憶管理
- タスク間通信

よって上の項目以外のカーネルの必要な機能はすべてユーザレベルで実現されている。Figure 1を参照。

システムコールのエミュレート、仮想記憶のためのメモリ管理ページャ、UNIX のファイルシステム、ネットワークなどはユーザレベルのタスクで実現されている。Figure 2を参照。

このように機能の分割により、ユーザは容易にカーネルの機能の変更、追加などが可能である。

Table 1: UNIX と Lisp Server の対応

UNIX	Lisp Server
page daemon	Garbage collection thread
init	init thread
getty	getty thread
csh	read,eval,print のループ
telnetd	ネットワークスレッド

3 Lisp Server の思想

Lisp Server は Micro Kernel Mach OS 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。

4 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

また Lisp Server は Unix と似ていて Table 1 のように対応している。Lisp Server の kernel の機能は以下のものがある。

- スレッドの管理

すべてのスレッド毎に必要なスタックなどの情報を管理する。ただし、スレッドの中

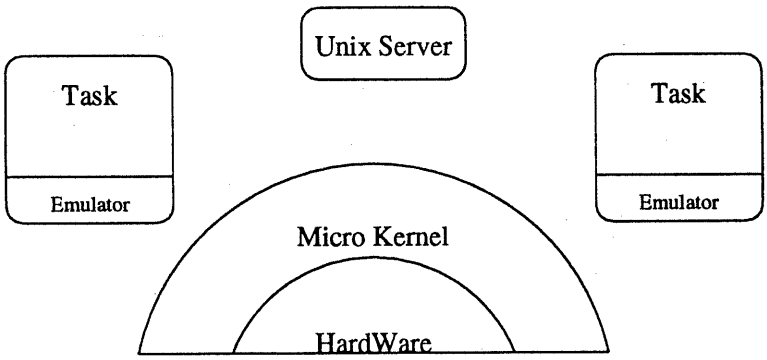

Figure 2: Mach OS と UNIX の構成

断、再起などはこのカーネル部分が行なうのではなく、スケジューラがすべてのスレッドのステータスを管理している。

- LISP のセル、アトム、フリーリストなどデータの管理
  すべての LISP のメモリつまり、セル、アトムなどの領域は Lisp Kernel の部分が管理を行なっている。これにより、並列処理、並列 GC などをサポートすることが可能である。
- 低レベルのデバイスドライバ
  コンソールなどのキーボードと画面のデバイスドライバをアクセス可能にして、システムコールスレッドからのデバイスドライバ要求の処理を行なう。

である。他の OS として必要な機能はすべて外部のスレッドとして実現する。

4.1 カーネル機能のスレッド化

Lisp Server のカーネルは以下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。

4.1.1 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。システムコールは以下のように分類できる。

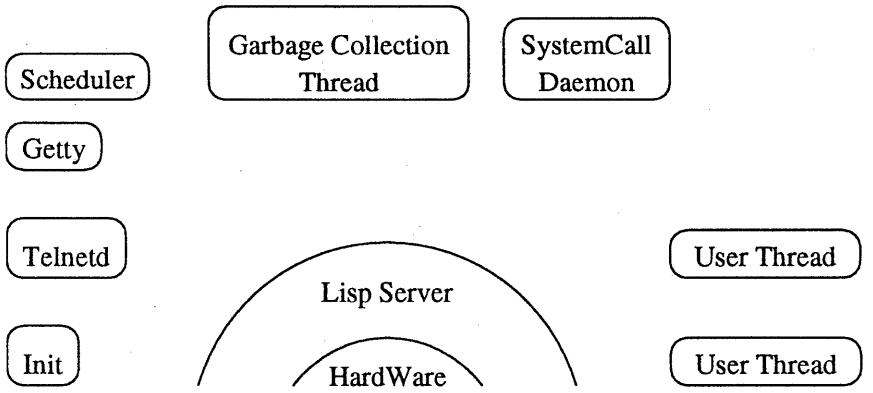

Figure 3: Lisp Server のシステムスレッド

- I/O
  デバイス, ネットワークの I/O をサポートしている。
- GC
  GC のプログラム 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。
- thread 間の排他制御
  thread 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。

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。

4.1.2 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。また、セル領域をある量以上を使うようなスレッドのプライオリティを下げるようにしている。現在はこのスレッドは 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用意している。

4.1.3 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。

4.1.4 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。

4.1.5 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。

4.1.6 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。

4.1.7 System call スレッド

これはすべてのスレッドからのシステムコール要求を受けつけその処理を行なう。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行なっている。またファイルシステムについても同じであり、UNIX のファイルシステムに変換を行なってエミュレートしている。

5 インプリメント

Lisp Server を現在 SunOS4.1.2[2]上でインプリメントされている。SunOS の Light Weigth Process(LWP) のライブラリを使っている。よって SunOS 上では Figure 4のような構造になっている。

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。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 の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。

[2]SunOS is trademark for Sunmicro System

6 Light Weight Process ライブラリ

SunOS の LWP ライブラリは以下の特徴を持っている。

- LWP のスケジューリングとプライオリティ
- LWP の context switching
- LWP の event mapping(agent)
- LWP 間のメッセージ通信
- LWP 間の同期

以上のようにライブラリを分けることができる。それぞれについてこの Lisp Server で使っているライブラリについて説明する。

6.1 LWP のスケジューリングとプライオリティ

LWP のスケジュールのアクセスのライブラリはスケジューリングのために使うことが可能

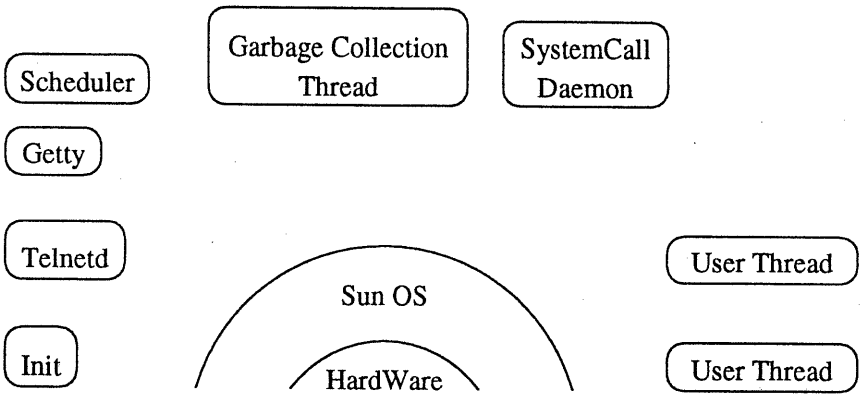

Figure 4: SunOS 上でのインプリメント

である。
以下の機能をもったライブラリがある。

- LWP の生成
- LWP の suspend
- LWP の resume
- LWP の terminate
- LWP の指定時間の sleep
- 複数の LWP の同じプライオリティのスケジュールのし直し。
- LWP のプライオリティ設定と変更

このライブラリを組み合わせることで簡単なスケジューラを作成することが可能である。例えばスケジューラの thread のプライオリティだけを大きくして他の thread のプライオリティを下げることによってスケジューラスレッドがある時間スリープすることにより、1つのスレッドが実行して、スケジューラスレッドのスリープした時間だけ経ったときにスケジューラスレッドが resume される。この後で、スケジューリングのし直しをおこなうことでスレッドを簡単にスケジューラーをプログラムすることが可能でである。

6.2 LWP の context switching

LWP 間の context switching の時に呼び出されるイベントハンドラを登録することが可能である。これにより、例えば個々のスレッドの実行時間を計測することが可能である。またスレッド毎に必要なデータをこのイベントハンドラが退避、復帰させることで容易にインプリメント可能である。このようなイベントハンドラがなければ常に現在実行中のスレッドの識別子をグローバル変数で持っているか、スケジューラスレッドが変数の退避、復帰をさせなければならない。例えば C 言語のグローバル変数の errno の値は普通はすべての LWP で共有しているのでどの LWP がこの変数に値をセットしかどうかがわからないが、このように errno をスレッドごとに退避させることができるので有

効である。

6.3 LWP の event mapping

event mapping は LWP では agent の機能である。agent の機能は signal を割り込みルーチンを LWP にすることが可能である。つまり signal が送られたときに LWP が resume させることが可能である。これによりシグナルを待っているスレッドに例えば SIGIO の I/O 関係のシグナルが発生した時にこのスレッドが resume されて、I/O の処理を行なように設定できる。これにより、常にループにより、I/O の要求があるかどうかを調べるのではなく、必要な時にのみこのスレッドに実行が移るようになる。

6.4 LWP 間のメッセージ通信

LWP 間のメッセージ通信のライブラリがあり、Lisp Server の場合はスレッドからのシステムコールをシステムコールスレッドに送るときに使っている。これにより、複数のスレッドからのメッセージを受けとることが可能で、しかもシーケンシャルに取り出すことが可能である。

6.5 LWP 間の同期

スレッド間の同期の変数などがあるが、このシステムではすべて実行しているスレッドの管理を行なっているためこの変数を使う意味がないので使っていない。

7 今後の課題

現在は LWP ライブラリを使ってインプリメントを行なっているが、これを Mach OS 上へのインプリメントを行ない、複数の CPU を持った MachOS の稼働する WS を使ってパフォーマンスを計測する。またスケジューラ、GC の部分を変更によるパフォーマンスへの影響を調査して、Lisp Server における最適なスケジューリング、GC の方法を検証する。

参考文献

[1] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, and Richard Sanzi. Mach: A foundation for open systems. *Proceedings of the Second Workshop on Workstation Operating Systems(WWOS2)*, September 1989.

[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 Internation Conference COMPCON 89*, February 1989.