システムソフトウェアと
オペレーティング・システム 67－3
(1994. 12. 8)

並列OS "K1" の実装と性能評価

原野裕樹†, 桑山雅行‡, 最所圭三†, 福田晃†

†奈良先端科学技術大学院大学情報科学研究科
‡九州大学大学院工学研究科情報工学専攻

奈良先端科学技術大学院大学情報科学研究科
630-01 生駒市高山町 8916-5

あらまし　並列オペレーティングシステム "K1" は、(1) ユーザに並列実行環境を提供すること、(2)OS 内部の処理を並列に実行すること、の2点を目標としている。

K1 では、(1) を実現するためにスレッドの概念を導入し、(2) を実現するためにマイクロカーネル構成にして、共有メモリ上に用意したメッセージプールを用いてプロセス間通信を行なう。OS への処理の依頼はメッセージプールを介してメッセージを送ることによって行なう。メッセージプールを通して処理の依頼を行なうことによって、不特定のアイドルプロセッサに処理を依頼することが可能となる。

我々は K1 をマルチプロセッサ構成のワークステーション BE 上に実現した。本稿では、K1 の概要、実装を述べ、性能評価を行なう。

和文キーワード　並列OS、メッセージプール、マイクロカーネル、性能評価

Implementation and Performance Evaluation of the K1 Multiprocessor Operating System

Yuuki HARANO†, Masayuki KUWAYAMA‡, Keizo SAISHO†, and Akira FUKUDA†

†Graduate School of Informatin Science, Nara Institute of Science and Technology
‡Department of Computer Science and Communication Enginnering, Kyushu University

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara 630-01 Japan

Abstract

The K1 multiprocessor operating system has the following two goals: (1) to give programmers parallel programming environments, and (2) to execute internal operations in parallel in an operating system.

For the goal (1), we introduce the process-thread model. For the goal (2), K1 is designed as micro-kernel fashion, and provides interprocess communication mechanism using a message pool implemented in a shared memory. Processes send request messages to OS activities through a message pool. That allows the activities to be executed on any idle processor.

We implemented K1 on a multi-processor workstation, the BE workstation. In this paper, we describe the concept, the implementation, and performance evaluation of K1.

英文 key words　multiprocessor operating system, message pool, micro-kernel, performance evaluation

1 はじめに

マルチプロセッサシステムの開発、実用化が行なわれて久しいが、マルチプロセッサシステムの性能は、プロセッサの結合形態とオペレーティングシステム (OS) との親和性に大きく依存している。しかし、現在マルチプロセッサシステム上で実用化されている OS の多くは、既存のシングルプロセッサシステム用の OS を複数のプロセッサ上で実行できるように拡張しているに過ぎず、マルチプロセッサシステムの資源を効率良く利用しているとは言えない。また、今後出現するであろう超並列計算機時代に対応するためにも効率の良い OS の開発が必要である。

我々は並列 OS"K1"[1][2][4][5][6] を、現在もっとも一般的なマルチプロセッサシステムである共有メモリ型マルチプロセッサシステムを対象として設計・開発を行なっている。K1 は、

(1) ユーザに並列実行環境の提供

(2) OS 内部の処理の並列実行

の 2 点に主眼をおいて設計、開発している。

(1) を実現するためにプロセス・スレッドモデルを提供し、(2) を実現するために、メッセージプール機構を導入した。OS への処理の依頼はメッセージプールにメッセージを送ることによって行なう。これによって、不特定のアイドルプロセッサに処理を依頼することが可能になる。

我々は、K1 の第 1 バージョンをマルチプロセッサのワークステーション BE 上に実現した。本稿では、K1 の概要、実装、性能評価について述べる。

2 K1 の概要

2.1 対象

将来のマルチプロセッサシステムは共有メモリ型のマルチプロセッサを 1 つのクラスタとし、クラスタ間を疎結合したシステムが主流になると考えている。K1 は、このクラスタ内の密結合システムを対象としている。

2.2 要件と設計方針

- ユーザへの並列処理環境の提供

  ユーザがアプリケーションの並列性を容易に引き出せるようにする。協調動作を行なう場合、その処理をプロセスに分割するとプロセス間の通信やメモリ空間の切替えが頻繁に起きるため、オーバーヘッドが大きくなる。このオーバヘッドを小さくするために、Mach 等と同様にプロセス内部をスレッドに分割して実行する、プロセス・スレッドモデルを提供する。

- OS 内部の並列化

  アプリケーションを並列に実行できても、OS が管理する資源を利用する場合に、OS の実行が逐次的であるとこの部分がボトルネックになる。そこで、OS 内部の処理をできるだけ並列に実行できるようにする。このため、マイクロカーネル構成にし、各サーバなどの間の通信はプロセスはメッセージプール機構を用いて行なう。

- スケーラビリティ

  将来も同一の OS でサポート可能にするには、OS がプロセッサ台数の変化に対応できなければならない。つまり、プロセッサの多いマシンで動かした時にプロセッサ台数に応じた処理速度が得られなければならない。このため、OS 内部の並列度を上げるだけでなく、ロックの数が多くならないようにしなければならない。

2.3 K1 の構成

K1 はマイクロカーネルと複数のサーバプロセスによって構成される。図 1 に、K1 の構成を示す。

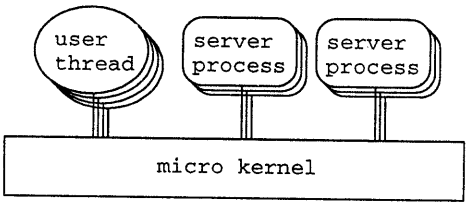

図 1: K1 の構成

3 メッセージプール機構

3.1 メッセージプール機構の概要

K1 では、プロセスやスレッド間の通信手段として、メッセージプールを介したメッセージ通信を提供し、K1 では UNIX などにおけるシステムコールをこのメッセージ通信を用いて実現している。

メッセージプールは、カーネル内に複数用意したメッセージポストからなり、ポストにはメッセージを接続するメッセージキューと、受信者を接続するレシーバキューが用意されている (図 2)。そして、メッセージのすべての送受信はポストに対して行なう。それぞれのキューに複数のメッセージやスレッドを接続することによって、1 対多、多対 1 などの任意のメッセージ通信が可能になる。

3.2 システムコール

以下にメッセージプール機構に関するシステムコールを説明する。これらのシステムコールは従来の UNIX などと同様に割り込みを用いて実行される。

- msg_get_post(post)

  番号 post のポストを獲得する。どのポストでも良い時は引数に−1 を与える。

- msg_free_post(post, flag)

  番号 post のポストを解放する。メッセージキューやレシーバキューが空でない場合は、flag が 0 でない時にはキューを空にした後に解放するが、flag が 0 である時はエラーを返す。

- msg_msend(post, msg, len, num)

  番号 post のポストに msg, len で示されるメッセージを num 個接続する。msg, len はそれぞれメッセージへのポインタと長さである。

- msg_receive(post, msg, len)

  番号 post のポストから msg, len で示される場所にメッセージを受け取る。len がメッセージの長さより長い場合は正常に受け取るが、短い時はエラーを返す。

- msg_sendrec(s_post, msg, len, r_post)

  番号 s_post のポストに msg, len で示されるメッセージを接続し、番号 r_post のポストからメッセージを受け取る。msg_msend と msg_receive をアトミックに実行したい時に使用する。

3.3 メッセージプール機構による負荷分散

K1 は、マイクロカーネル化によって、機能がマイクロカーネルとサーバプロセス群に分かれており、ユーザがシステムコールを発行する時にはメッセージプール機構を用いてサーバプロセスに処理を依頼する。

メッセージプールは機能別に分けられており、処理を依頼する場合、その機能に対応するポストにメッセージを接続する。各サーバはあらかじめ受信状態になって待っており、メッセージの到着によってレディキューにつながれ、アイドルプロセッサによって実行される。

しかし、5.4節で述べるように、msg_sendrec の場合には、メッセージの接続によってサーバを実行しようとしたプロセッサとカーネル内の資源の競合を起こすので、メッセージを接続した時にアイドルプロセッサがサーバの実行を開始しないようにしなければならない。

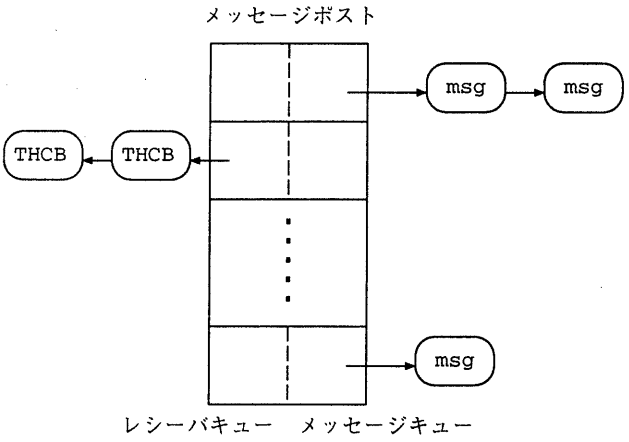

図 2: メッセージプールの概念図

4 実装

4.1 対象

4.1.1 ハードウェア構成

K1 のターゲットシステムである Panasonic 社製のワークステーション BE [7] のハードウェア構成について説明する。

BE には CPU として Intel 社の i386 (25MHz) を用いており、拡張ボードに CPU ボードを 3 台まで増設できる。メモリはすべての CPU で共有しており、各 CPU 毎のローカルなメモリは持たない (図 3)。すべての CPU とメモリ及び周辺機器制御回路は 1 本のバスでつながれており、すべての CPU からアクセスできる。ただし、割り込みを受ける CPU は 1 度に 1 つであり、どの CPU が割り込みを受けるかはプログラムで制御できる。

また、CPU 間の通信は割り込み (NMI) とメモリを用いて行なう。

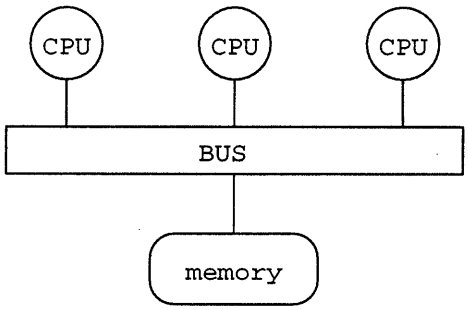

図 3: BE のメモリ形態

4.1.2 開発環境

K1 の開発環境は、図 4 に示すように BE を 2 台使用し、2 台の BE をファイル転送用のイーサネットで接続している。1 台はホス

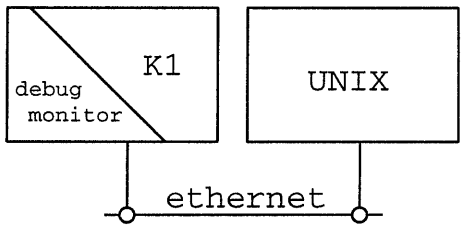

図 4: 開発環境

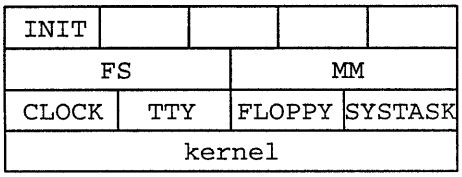

図 5: MINIX の構成

トマシンとして UNIX を動作させ、プログラムの作成とコンパイルを行なう。もう 1 台はターゲットマシンとして CPU を 3 台に増設しており、簡単なモニタプログラムを動作させている。このモニタプログラムは、ホストのハードディスクからイーサネットを通して実行形式のプログラムをロード実行させることができる。

現在、K1 は 8 つの実行形式プログラムに分かれており、モニタプログラムは、8 つのプログラムをロードし、カーネルの先頭アドレスに制御を移す。

4.2 実現方法

K1 を実現するのに初めから作成するのは困難であるため、K1 と構成が似ている MINIX をベースにして K1 を実現することにした。ここで、MINIX を選択したのは次のような理由からである。

- MINIX は図 5 に示すように、マイクロカーネル化されている[1][8]。
- プロセス間通信にメッセージを用いている。MINIX のメッセージ通信はランデブー方式であるが、大幅な変更は少ない。
- 比較的小さく、全ソースコードを入手できる。

4.3 概要

以下に実現した機能について述べる。

[1]MINIX ではマイクロカーネルとは言っていない。

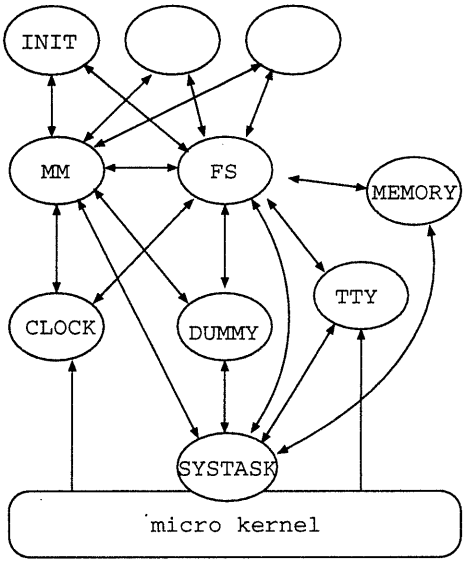

図 6: 実際の K1 の構成

- 共有メモリ型マルチプロセッサシステム

  カーネルに一度に 1 つの CPU しか入れないようにする、実行中のプロセスをレディキューからはずす、などの変更を加え、共有メモリ型のマルチプロセッサシステムに対応した。

- マイクロカーネル構成

  図 6 に示すように、マイクロカーネル、デバイスドライバ群、メモリマネージャ、ファイルシステムに分離している。図中の矢印はメッセージの送受信関係を示す。

  また、現バージョンでは MINIX の影響で、マイクロカーネルにプロセス管理機能が含まれている。このため、カーネルが持つプロセステーブルを変更するために SYSTASK というタスクが存在している。マイクロカーネルにプロセスやスレッドの概念を含めるべきでないので [3]、将来的には、SYSTASK をプロセスマネージャとして独立させる予定である。

- メッセージプール機構

  メッセージプール機構のほとんどを実現している。しかし、重要な機能の1つであるポストの保護機能が未実装であるため、ユーザプロセスが一部のサーバプロセスに間違ったメッセージを送信すると、OS 全体が停止することがある。保護機能に関しては現在検討中である。

  また、ユーザプロセスがメッセージを大量に送信したまま受信されないと、メッセージプールが溢れ、ハードウェア割込時にデバイスドライバにメッセージが送信できないことが考えられる。このため、一部をサーバプロセス用に予約した。

  現段階では、msg_sendrec の時にもアイドルプロセッサがサーバを実行しようとするので、3.3節で述べた競合が起きることがある。

- プロセス・スレッドモデル

  現在、スレッドはサポートしていない。

  また、プロセスを生成する fork システムコールをメッセージ通信によって実現するため、子プロセスにもシステムコールの結果を返す。

- 割り込み

  ハードウェア割り込みが発生した時には、カーネル内で必要最小限の処理を行ない、対応するデバイスドライバにメッセージを送信し、メッセージを受け取ったデバイスドライバが実質的な処理を行なう。

  また、アイドルな CPU が割り込みを受けるのが理想的だが、BE のハードウェアの制限により、アイドルでない CPU が割り込みを受けることもある。

- デバイスドライバ群

  MINIX を改造する方法で作成したのでほとんどの部分は変更しないで実行できる。しかし、MINIX ではルートデバイスが必ず必要にも関わらず、現在はフロッピーディスクもハードディスクもサポートしてないので、新しいブロック型デバイス DUMMY を用意した。このデバイスは RAMDISK に非常に似ているが、容量は 360KB で、コンソールデバイスのみを有し、読みとり専用である。

	送信する	送信しない
MINIX	148	148
K1/sp	385	381
K1/mp	489	482

表 1: 測定結果 (単位:1/60 秒)

5 性能評価

5.1 測定

まだディスク類をサポートしていないので、コマンドを実行させての評価はできない。このため、OS 内にテストプログラムを埋め込んで実行する方法により測定を行なった。

実行したテストプログラムは、init プロセスからの合図で、ファイルマネージャとメモリマネージャの間でメッセージのやりとりを10000往復行なうものである。タイマの割り込みで CLOCK にメッセージを送信する時としない時で、MINIX と K1 のシングルプロセッサバージョン (以下、K1/sp) と K1 のマルチプロセッサバージョン (以下、K1/mp) を比較し、メッセージプール機構、マイクロカーネル構成、マルチプロセッサの影響について調べた。

ただし、K1/sp と K1/mp の違いは、K1/sp が CPU を1つしか動かしていないことだけであり、排他制御のためのロック等は行なっている。これに対して、MINIX ではソフトウェアによるロックは行なっていない。実行結果を表1に示す。

5.2 メッセージプール機構の影響

メッセージプール機構の影響を調べるためにMINIXとK1/spを比較するが、タイマ割り込みによってCLOCKにメッセージを送るとマイクロカーネル化が影響するので、CLOCKにメッセージを送らない状態で比較する。

表1から、K1/spはMINIXの2.57倍の時間がかかっている。これは次の理由からだと考えられる。

- メッセージが28バイトから48バイトに大きくなった。
- メッセージのコピーを2回行なうため、オーバーヘッドが大きい。

5.3 マイクロカーネル構成の影響

マイクロカーネル構成の影響を調べるために、タイマ割り込みによってCLOCKにメッセージを送る時と送らない時で、MINIXとK1/spの差の変化を調べる。

表1から、MINIXでは変化がないのに比べて、K1/spでは少しだが長くかかっている。つまり、差が広がっている。これには次のような理由が考えられる。

- CLOCKタスクにメッセージを送信するのにメッセージプール機構を用いているが、このメッセージプール機構のオーバヘッドが大きいことが影響している。
- MINIXではCLOCKタスクがカーネルと同じ空間にあるのに対して、マイクロカーネル構成ではCLOCKプロセスの空間に切替えるために、メモリキャッシュが有効に使われていない。

5.4 マルチプロセッサの影響

マルチプロセッサの影響を調べるためにK1/spとK1/mpを比較する。タイマ割り込みによってCLOCKにメッセージを送るとCLOCKがレディ状態になる。この時、K1/spではテストプログラムを実行中のCPUがCLOCKを実行するのに対し、K1/mpではアイドルCPUがCLOCKを実行する。従って、CLOCKにメッセージを送信する状態でK1/spとK1/mpを比較することによって、マルチプロセッサの影響を調べることができる。

表1から、K1/mpの方が少し長くかかっている。これは、3.3節で述べた競合が起きているためである。詳しくは、次のような状況が起きている。

1. CPU1がプロセスAを実行している。
2. プロセスAがプロセスBにメッセージを送信するためにシステムコールを発行する。
3. CPU1がカーネル内に入る。
4. プロセスBがレディ状態になる。
5. CPU2がプロセスBを実行するためにカーネルに入ろうとする。この段階ではCPU1がまだカーネルにいるため、入れずに待つ。
6. プロセスAは送信直後に受信のためにサスペンドされる。
7. CPU1は別のレディ状態のプロセスBの実行を開始し、カーネルから出る。
8. CPU2がカーネルに入り、レディ状態のプロセスを実行しようとする。
9. プロセスBがプロセスAにメッセージを送信するためにシステムコールを発行する。
10. CPU1がカーネルに入ろうとするが、CPU2がカーネルにいるため待つ。
11. CPU2がカーネルから出て、再びレディ状態のプロセスができるまで待つ。
12. CPU1がカーネルに入り、プロセスAを起こす。

13. プロセスAが実行され、メッセージをプロセスBに送信すると、3以下が繰り返される。

つまり、CPU1の処理をCPU2が邪魔しているのである。テストプログラムがシステムコールの発行しか行なっていないために待機が頻発しているが、実際の処理では問題にならないだろう。しかし、今後、プロセッサの多いマシンに移植した時に問題となる可能性がある。

6 おわりに

K1の概要、実装、性能評価について述べた。現在、スレッドをサポートしておらず、デバイスドライバは少ない。今後、これらの機能を実現し、効率的なアプリケーションの開発や、アプリケーションレベルでの性能評価が必要である。当初から予想されたことではあるが、メッセージ機構がシステム性能に大きな影響を与える。今後、一般的なアプリケーションを用いて、メッセージ送受信頻度などの測定を行ない、これらを通してメッセージプール機構の改良を行なっていく必要があると思われる。

また、BEはプロセッサが少ないために競合があまり問題になっていないが、プロセッサの多いマシンに移植し、どの程度問題になるのかを調べる必要がある。

さらに、K1の対象は共有メモリ型のマルチプロセッサシステムであるが、K1を拡張して、メッセージパッシング型のマルチプロセッサシステム上でのメッセージ通信を実現し、メッセージパッシング型と共有メモリ型のマルチプロセッサシステムの性能の評価、検討を行なうことによって、より広い応用が可能となると考えている。

参考文献

[1] 宮崎輝樹, 桑山雅行, 最所圭三, 福田晃: "並列オペレーティング・システムK1 –メッセージプール機構–", 情報処理学会オペレーティング・システム研究会, 56-4 (1992).

[2] 今村信貴, 桑山雅行, 宮崎輝樹, 林茂昭, 福田晃, 富田眞治: "並列オペレーティング・システムK1の設計と実現 –フリー・プロセッサ・キューを用いたプロセッサ管理–", 並列処理シンポジウムJSPP '92, pp.305-312 (1992).

[3] 桑山雅行, 最所圭三, 福田晃: "並列オペレーティング・システムK1 –マイクロカーネルの考察と設計–", 情報処理学会「コンピュータ・システム・シンポジウム」論文集, Vol.92, No.7, pp.69-76 (1992).

[4] Kunihiko Tsunedomi, Akira Fukuda, Kazuaki Murakami, Shinji Tomita: "A Message-Pool-Based Parallel Operating System for the Kyushu University Reconfigurable Parallel Processor – Parallel Creation of Multiple Threads–", J. of Information Processing, Vol.14, No.4, pp.423-432 (1992).

[5] 今村信貴: "並列オペレーティングシステムK1の基本設計 –プロセッサ管理–", 九州大学大学院総合理工学研究科情報システム学専攻修士論文 (1992).

[6] 宮崎輝樹: "並列オペレーティングシステムK1第1版の実現 –メッセージプール機構とプロセス管理–", 九州大学工学部情報工学科卒業論文 (1992).

[7] 松下コンピュータシステム株式会社: "ハードウエア解説書 BE シリーズ", (1989).

[8] Andrew S.Tanenbaum: "MINIX OPERATING SYSTEMS Design and Implementation", Prentice-Hall (1987).