オペレーティング・システム 56－4
(1992. 8. 19)

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―

宮崎 輝樹† 桑山 雅行‡ 最所 圭三* 福田 晃†

† 九州大学工学部情報工学科
‡ 九州大学大学院総合理工学研究科
* 九州大学工学部中央計数施設

共有メモリ型マルチプロセッサを対象とした並列オペレーティング・システム "K1" は，OS内部の負荷分散を1つの目的としている．K1はメッセージプールをマイクロカーネルとし，その上に各種サーバがのる構成をとる．各サーバはメッセージプールを介して通信する．メッセージプールは以下の特徴を有している．

1) 各サーバに対して統一的な通信の場を与える．
2) 各サーバの負荷を分散できる機構を提供する．
3) メッセージプールをマイクロカーネルにすることにより，マイクロカーネルの利点を引き継ぐことができる．

本稿では，メッセージプール機構の機能について述べる．さらに，WS"BE" 上への実装状況を報告する．

The K1 Parallel Operating System -Message-Pool System -

Teruki Miyazaki† Masayuki Kuwayama‡ Keizo Saisho* Akira Fukuda†

† Department of Computer Science and Communication Engineering,
Faculty of Engineering,Kyushu University
‡ Interdisciplinary Graduate School of Engineering Sciences,
Kyushu University
* University Computation Center,Faculty of Engineering,
Kyushu University
Correspondence: {teruki, kuwayama, fukuda}@csce.kyushu-u.ac.jp,
sai@keisu.kyushu-u.ac.jp

The K1 parallel operating system aims at parallel processing in operating system itself. K1 kernel consists of message-pool system, which allows multiple servers on the system to communicate together. The message-pool system has the following features:

1) It gives an uniform communication method.
2) It gives mechanism which allows loads of servers to be balanced.
3) It has the same features as micro-kernel when it is micro-kernelized.

This paper presents functions of the message-pool system, and also reports the current states of the implementation of the system on the BE workstation.

1 はじめに

現在，計算機の高速化を目的としたさまざまな研究がなされているが，中でもマルチプロセッサシステムの開発が盛んである．マルチプロセッサシステムを有効に利用するためには，マルチプロセッサの能力を引き出せるようなオペレーティングシステム (以下 OS) の研究が必要である．そこで我々は，共有メモリ型マルチプロセッサシステムを対象とする OS"K1" [1][2][3][4][5]を研究開発している．

K1 を開発するにあたり我々は，(1)ユーザへの並列処理環境の提供，(2)OS 内部処理の並列化，の 2 点を要件としてあげた．そして，これらの要件を満たすために，スレッドの概念を導入し，またメッセージプール機構を用いた負荷分散方式を採用した．

メッセージプール機構とは共有メモリ上のメッセージプールを通信場とするメッセージ通信機構である．我々は K1 を，メッセージプール機構からなるマイクロカーネルを中心として 2 つのマネージャ，およびデバイスドライバ群に機能分割した．ユーザースレッドを含むこれらのスレッド群は，互いにメッセージプール機構を用いたメッセージ通信で，処理を依頼したり，処理を受けたりする．それにより，不特定のアイドルプロセッサに負荷を分散させることができる．また，これにより，マルチプロセッサシステム上の全てのプロセッサを均一に見ることができるので，OS の構築が容易となり，プロセッサ数の台数の変化にも対応しやすい．

本稿では，K1 でメッセージ通信機構として採用したメッセージプール機構の機能について述べ，メッセージプール機構の WS"BE" への実装方法について説明する．また，現在のメッセージプール機構の欠点とその解決方法について考察する．

2 K1 の設計方針

2.1 対象ハードウェア

マルチプロセッサシステムは，プロセッサ間の結合によって共有メモリ型とプライベートメモリのみからなるメッセージパッシング型の 2 種類に大きく分類できる．我々は，将来のマルチプロセッサシステムは共有メモリの MIMD 型マルチプロセッサを 1 クラスタとし，クラスタ間はメッセージパッシング方式のシステムが主流になると考えている．そこで K1 では第一段階としてこのクラスタ内の共有メモリ型システムを対象とする．ただし，クラスタ内であっても，均一メモリアクセス (UMA:Uniform Memory Access) アーキテクチャのままで台数を増加させるのは難しく，将来的には不均一メモリアクセス (NUMA:Non-Uniform Memory Access) アーキテクチャのシステムが増えると考えられる．従って K1 では，UMA アーキテクチャと NUMA アーキテクチャの両方を対象とする．

2.2 要件

マルチプロセッサシステムを対象とする OS では，アプリケーションや，OS から出来るだけ並列性を引き出し，並列性を損なわぬように実行することが重要である．このため，以下のような要件を満たす必要がある．

(1) 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すことができ，マルチプロセッサシステム上で，効率的に並列実行できるプログラミング環境を用意する必要がある．

(2) OS 内部処理の並列化

アプリケーションの並列性を抽出しても，OS の実行が並列にできないと結果的に並列に処理されない場合が出てくる．そこで，OS 内部の処理をできるだけ並列に実行可能にし，プロセッサ資源を有効利用する必要がある．

一般に並列化には，対象とする処理内容にもよるが，1 つの仕事を複数の異なる機能に

分割して並列実行させるコントロールパラレル化と，取り扱うデータを分割して同一の機能を並列実行させるデータパラレル化の2つがある．これまでに，異なる機能を並列に実行するコントロールパラレル化は一部のOSで実現されている[6][7]．K1では，これに加えてOS内のデータパラレル化を行なうことを目的とする．すなわち，オーバーヘッドの問題に対処する必要があるが，同一の機能を機能分割およびデータ分割して，OS内部の並列処理を実現する．

2.3 設計方針

前節で述べたような要件を満たすために，K1では以下のような設計方針をとった．

(1) プロセス・スレッドモデルの提供

Mach[8]等と同様にプロセス内部を並列実行可能な，処理の流れであるスレッドの概念を導入し，実行単位の生成や並列実行時の同期や通信によるオーバーヘッドを抑える．

(2) メッセージプール機構による処理の並列化

OS内部の並列化を実現する方法として，共有メモリ上のメッセージプールと呼ばれる通信場を介して負荷分散を行なうOSを構築する．メッセージプール機構の実際の機能については次章で述べる．

2.4 K1の構成

K1は，メッセージプール機構を含むマイクロカーネルと，プロセスマネージャとファイルマネージャの2つのマネージャ，およびデバイスドライバ群からなる(図1参照)．これらの機能別に分けた各部分が，カーネル内のメッセージプール機構によって相互に通信し，処理を行なう．以下に各部分の機能について述べる．

(1) マイクロカーネル

マイクロカーネルは，メッセージプール機構，コンテキストスイッチ機構，および割り込みハンドラからなる．マイクロカーネルだけは，割り込みをかけることによって実行される．また，ハードウェアからの割り込みが起きた場合，カーネル内でメッセージを生成し，指定のデバイスドライバに送信する．

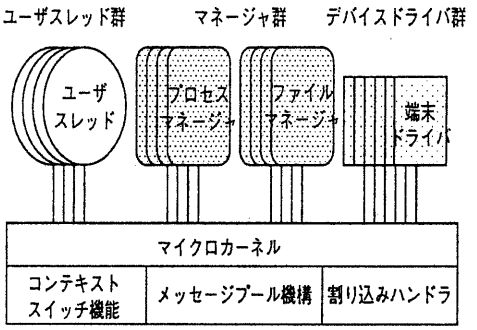

図1 K1の構成

(2) プロセスマネージャ

プロセスマネージャは，メモリ管理，シグナル管理，およびプロセス関連のシステムコールを実行する．プロセスマネージャおよびファイルマネージャは，カーネルとは完全に独立したプロセスとして実行される．また，起動時に複数のスレッドを生成しそれぞれを並列に実行可能とする．

(3) ファイルマネージャ

ファイルマネージャでは，ディレクトリの管理，バッファリング，ファイル関連のシステムコールを実行する．ファイルマネージャもプロセスマネージャと同様に，複数のスレッドによって並列に実行可能とする．

(4) デバイスドライバ群

デバイスドライバは，各デバイス毎に独立したプロセスとして用意する．

3 メッセージプール機構

3.1 特徴

(1) 各マネージャに対して統一的な通信の場を提供する

メッセージプール機構は，柔軟なメッセー

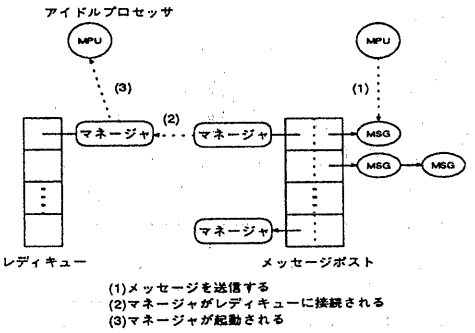

図2 メッセージプール機構を介した負荷分散

ジ通信形態を提供しており，それにより，各マネージャ間での通信が容易になる．例えば，あるマネージャからマネージャのスレッド群への1対多通信を用いることにより，複数スレッドの並列生成や，各プロセッサ間でのコヒーレンス制御等が簡単に実現できる．また同様に多対1通信は，複数スレッドの同期等に用いることができる．

(2) 負荷分散機能

各マネージャやデバイスドライバ群の基本的な構成は同じで，それらのスレッドは処理の依頼を受けるためメッセージ受信のシステムコールを発行して，メッセージ待ちの状態でサスペンドされている．ユーザまたは他のマネージャから処理を依頼するメッセージが送られると，メッセージ待ちをしていたマネージャは自動的にレディキューに接続され，並列実行可能であればアイドルプロセッサを起こして実行される．このようにして，不特定のアイドルプロセッサに対して負荷を分散させることができる．それにより，OSから全てのプロセッサを均一的に扱うことができ，OSの構築が容易となる(図2参照)．

(3) マイクロカーネル化の利点

また，メッセージプール機構を中心としてマイクロカーネルを構成したことにより，マイクロカーネルの特徴の一つである，OSのモジュール化や，階層化を図りやすく，OSの拡張も容易である[9][10]

3.2 メッセージプール機構の構造

K1ではメッセージプールを，多数のメッセージポスト(以下ポスト)の集まりで構成した．ポストとは，送信されたメッセージをキューイングするメッセージキューと，メッセージ待ちのスレッドを接続するレシーバキューからなる．メッセージの送受信は全てポストを獲得し，ポストに対して行なうものとする．

3.3 メッセージプール機構のプリミティブ

メッセージプール機構のプリミティブとして，以下のようなものを用意した．

(1) get_post()

ポストを獲得する．戻り値として，各ポストに付けられた一意の数であるポスト番号を返す．メッセージプールはシステムに1つ用意され，メッセージプールを構成するポストへのアクセスは，ポスト番号によって行なわれる．マネージャやデバイスドライバにはシステム起動時に特定のポストが割り当てられるため，get_postする必要がないが，ユーザプロセス間で通信をする場合のために用意した．

(2) free_post(post)

ポストを解放する．引数として，get_postで獲得したポスト番号を渡す．

(3) msg_send(&msg)

メッセージを送信する．引数として，メッセージへのポインタを渡す．送信するポストは，後で述べるメッセージのヘッダ部分に指定する．送信されたメッセージは，受信者が揃っていない場合，メッセージプール機構内のバッファにコピーされ，指定のポストのメッセージキューに接続される．

(4) msg_receive(&msg)

メッセージを受信し，指定したアドレス(ポインタ&msgによって示されるアドレス)にメッセージをコピーする．まだメッセージが届

いていない場合は，自分をレシーバーキューに接続してサスペンドする．サスペンドしたスレッドは，メッセージの送信者によって起こされる．メッセージを受けとった場合，ヘッダ内のカウンタを1つ減らし，カウンタが0になった場合に始めてメッセージをメッセージキューから外す．

3.4 メッセージのデータ構造

メッセージプール機構で送受信するメッセージは，固定長のヘッダ部分と可変長のデータ部分とに分けられる．ヘッダ部分には以下のような，メッセージプール機構で必要なデータを格納し，データ部分はユーザで自由に設定できる．

(1) ポスト番号

メッセージを送信する相手のポスト番号，またはメッセージを受信するポスト番号を指定する．全てのスレッドは，生成時に1つポストを獲得し，そのポスト番号はTHCB(THread Control Block) に格納される．メッセージプール機構が送信側から，受信側へヘッダーのコピーをする際には，ここには送信スレッドの持っているポスト番号を入れる．また，受信時にここへ0を入れると，そのスレッドの持っているポスト番号のポストに対してメッセージ受信の操作を行なう．

(2) データ部分へのポインタ

データ部分の先頭アドレスを格納する．データ部分は，ヘッダ部分とは独立しており，任意の大きさに設定できる．メッセージ受信を行なう場合は，メッセージのデータ部分を格納するバッファへのポインタを格納する．

(3) データ部分の大きさ

データ部分の大きさを，バイト単位で格納する．メッセージ受信の際は，受信スレッドが確保したバッファ領域の上限を入れる．もし，送信されたメッセージの大きさが，用意しているバッファの大きさより大きい場合，送信および受信のシステムコールは失敗する．

(4) メッセージのカウンタ

同じメッセージを複数の相手に送信する場合のために，メッセージを送信する数を格納する．通常1を指定するが，複数の値を入れることにより，メッセージのマルチキャストが可能となる．

(5) キューへのリンク情報

ポスト内のメッセージキューに接続する時のための，リンク情報を格納する．この領域はメッセージプール機構で使用する．この情報を読み込む場合や，変更する(メッセージの挿入，および削除)場合には，キューの整合性を保つために，ロックを獲得してから行なう．

4 実装

4.1 開発環境

現在我々は，K1をワークステーション(WS) "BE" 上に実装している．BEは3台のプロセッサ(i80386)と共有メモリをバスで結合した，マルチプロセッサシステムである．BEを2台用意し，1台をプログラムの開発，コンパイルなどをするホストマシンとして，もう1台をK1を実装するターゲットマシンとしている．

ターゲット上には，シングルプロセッサで動作する簡単なモニタが動作していて，ホストから実行形式ファイルをメモリ上に格納し，起動する．

4.2 現状

現在，K1はまだシングルプロセッサ環境下ではあるが，メッセージ通信機能と，プロセスマネージャの主な機能を開発した．ファイルマネージャはまだ実装されていない．但しこれらマネージャとマイクロカーネルはまだはっきりと分離されていない．また，デバイスドライバの構築が困難さをさけるため，その解決策として図3のような構成をとった．つまり，ターゲット上に仮想デバイスドライバ，ホスト上にデバイスモニタを作り，ターゲットとホストをつなぐシリアル回線で通信

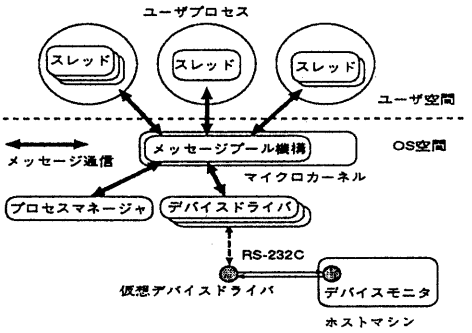

図 3 K1 の実装形態

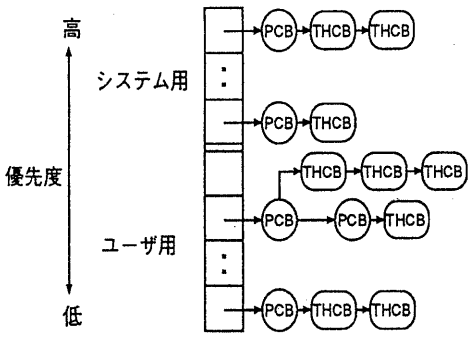

図 4 レディキュー

をして，ホスト上で，ターゲットの入出力デバイスをエミュレートする．ターゲット上でのデバイスへのアクセス要求は，仮想デバイスドライバにより，シリアル回線を用いたホストへのパケットに変換され送信される．これにより，ターゲット上で実装するデバイスドライバはシリアル回線ドライバだけですんだ．

4.3 プロセスマネージャ

現在実装しているプロセスマネージャの機能について説明する．プロセスマネージャで管理するデータ構造として主なものにPCB(Process Control Block) と THCB およびレディキューがある．

PCB には，プロセス内のスレッドが獲得した資源の管理のための情報や，スレッドを統一的に管理するための情報，また，実行可能状態のスレッドを接続するレディスレッドキューなどがある．THCB には，プロセッサのレジスタ情報等が格納されている．

K1 のレディキューは図 4のように 2 段階になっている．優先度はプロセス単位で計算しており，プロセス内のスレッドは FIFO となっている．

また，スレッドに関するシステムコールとして，次のようなものを用意した．

(1) th_sfork(n, func, arg)
スレッドを n 個生成して，関数 func を並列に実行する．th_sfork の動作を図 5に示す．このシステムコールを実行したスレッド (以下親スレッド) は，システムコール実行後システムコール以降の処理を続ける．生成されたスレッド (以下子スレッド) は，関数 func を実行し終えると，自動的に消滅する．arg は関数 func への引数である．

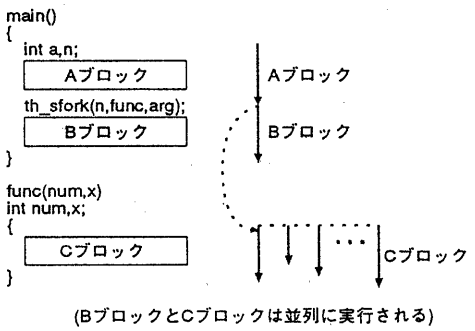

図 5 th_sfork の動作

複数生成した子スレッドの処理を区別するために，関数 func では，引数 arg とは別に，子スレッド間で一意な，0 から n-1 の値 (スレッドインデックス) を与えるようにした．このシステムコールは，メッセージ送信のマルチキャスト機能を使用して，並列に行なえる．

(2) th_exit()
スレッドを終了させる関数である．子スレッドは，関数 func を終了すると自動的に消滅

するが，途中で強制的に子スレッドを終了させる場合は，このシステムコールを用いる．

5 問題点と解決法

現在K1はまだ開発途中にあり，各種機能の性能を評価するに至っていない．しかし実際に実装することにより，不備な点も明らかになってきた．以下に問題点とその解決方法を示す．

現在のメッセージプール機構では，メッセージ送信時に送信スレッドからカーネル内のバッファへ，またメッセージ受信時にカーネル内のバッファから受信スレッドへと2回のコピーを行なっている．これは，OS内の処理を並列に実行するために送信側と受信側の同期をとらないようにしたためである．しかし，これではコピーによるオーバーヘッドが大きくなるものと思われるため，K1では，コピー・オン・ライト方式を用いることにした．

また，現在マネージャは2種類としているが，これは機能の分けやすさから2種類としているだけなので，K1の実装終了後，さまざまな分割単位で性能を評価し，マネージャの分割単位を決定する．

さらに，現在はメッセージプールは1つとなっており，ここがボトルネックになる可能性が大きい．そこで，メッセージプールの分散データ機能を検討する必要がある．

6 おわりに

K1でOSのプリミティブとして採用しているメッセージプール機構について述べ，K1の実現状況について説明した．今後メッセージプール機構を評価し，マルチプロセッサ対応へ拡張する必要がある．

謝辞 BEおよび開発環境を貸与して頂いている松下電器産業(株)映像音響研究センター・メディア研究所・システム第1開発室の浅原重夫氏(室長)，日高教行氏ならびに関係者の方々に感謝致します．

参考文献

[1] 恒富邦彦，福田晃，村上和彰，富田眞治: “可変構造型並列計算機のメッセージ指向型並列ＯＳとシミュレーションによるスレッド並列生成の評価”，コンピュータ・システム・シンポジウム，pp.67-74(1991).

[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－”，コンピュータ・システム・シンポジウム発表予定(1992).

[4] A.Fukuda, K.Murakami, and S.Tomita: “Toward Advanced Parallel Processing: Exploiting Parallelisms at Task and Instruction Levels”, IEEE Micro, Vol.11, No.8, pp.16-19, 50-61(1991).

[5] K.Tsunedomi, K.Murakami, A.Fukuda, and S.Tomita: “A Message-Pool-Based Parallel Operating System for the Kyushu University Reconfigurable Parallel Processor”, 情報処理学会欧文誌,Vol.14, No.4,pp.423-432(1992).

[6] 田胡和哉，益田隆司: “オペレーティング・システムの構造記述に関する一試み”，情報処理学会論文誌,Vol.25, No.4, pp.524-534(1984).

[7] 中山泰一，田胡和哉，森下巖: “プロセスネットワークとして実現したUNIXカーネルの並列動作によるシステム・コール・レスポンス時間短縮の試み”，情報処理学会論文誌，Vol.33, No.3, pp.330-337(1992).

[8] M.Accetta et al.: “Mach : A New Kernel Foundation for UNIX Develop-

ment,", Proc. USENIX 1986 Summer Conference,pp.93-112(1986).

[9] 前川守，所真理雄，清水謙太郎: "分散オペレーティングシステム－UNIXの次にくるもの"，共立(1991).

[10] M.Rozier, V.Abrossimov. F.Armand, I.Boule,
M.Gien, M.Guillemont, F.Herrmann, C.Kaiser, S.Langlois, P.Leonard, and W.Neuhauser: "Chorus Distributed Operating Systems", Computing Systems, Vol.1, No.4(1988).