システムソフトウェアと
オペレーティング・システム 87－6
（２００１．６．２８）

実時間制御システム用のマルチサーバ型分散OSの検討

日高 宗一郎　児玉 和也　丸山 勝巳
E-mail: {hidaka,kazuya,maruyama}@nii.ac.jp
国立情報学研究所

組込システム用 OS の需要は、実時間制御システムから情報家電まであるがその要求条件は適用対象により大いに異なる。本稿ではこのような要求条件を満たすために必要な OS の諸性質についての検討を行ない、最新のマイクロカーネル技術とマルチサーバ方式によりそのような OS が構成出来ることを主張している。検討中の OS は多様な要求に応えられる拡張性、高信頼性、高度な分散処理をサポートする事が期待される。現在 L4 マイクロカーネル上にマルチサーバ OS である Minix の移植を進めながら割り込み処理と保護機能の問題、論理空間切り替えの効率の問題、分散処理のために必要な枠組について検討している。

A study on multi-server distributed OS for real-time systems

Soichiro Hidaka, Kazuya Kodama and Katsumi Maruyama
E-mail: {hidaka,kazuya,maruyama}@nii.ac.jp
National Institute of Informatics(NII)

Demands of operating systems for control systems ranges from real-time systems to intelligent home appliance. However, what is needed depends on their applications. This paper discusses OS characteristics desired to meet above demands and claims that such OS consists of multi servers running on state-of-the-art μ-kernel. Our OS is expected to support extensibility, reliability, and highly-distributed processing functionalities. Porting of Minix multi-server OS on L4 μ-kernel is underway. Issues such as interrupt handling and protection, switching of address space, efficiency aspects, framework to achieve distributed processing are also described.

1 はじめに

プロセス制御システムや公衆通信網の交換システムのような大規模実時間システムから、情報家電のような小型実時間システムに至るまで、OS は制御システム構築の要である。このような制御システムは、適用分野毎に特別な機能や性能が要求される。例えば超多重処理・実時間性能・高信頼性、融通性の高い分散処理機能である。特に要求条件が厳しいシステムの場合には独自 OS を開発し、要求条件が緩い分野ではモニター的簡易 OS が使われているのが現状である。また、OS 研究はソフトウェア研究の基盤でありながら、日本では研究者が少なく研究強化が必要である。

以上の観点から, 我々は以下を特徴とする OS の研究を開始している。

(1) マイクロカーネル技術とマルチサーバ OS 技術により、機能拡張や独自機能の追加のを容易化、性能の Tuning 等を可能とする。

(2) OS による分散オブジェクトのサポートにより、高度で融通性に富んだ分散処理を容易に実現できるようにする。

(3) 強固なガード機構による信頼性の強化

(4) 巨大ストリームデータのサービス品質保証

本稿では、第 2節で適応型分散 OS の必要性と要求条件について述べ、それに対する本研究のアプローチについて第 3節で述べる。次にその要素である L4、マルチサーバ構成、分散処理機能についてそれぞれ 4、5、6節で述べる。

2 適応型分散 OS の必要性と要求条件

制御システム用 OS に要求される条件には、一般に以下のようなものがある。

(1) 適用分野の要求に応じて、機能の追加・変更が容易に行えること。

(2) 性能のチューニングが可能なこと。例えば、ユーザ独自のスケジューリングアルゴリズムやメモリ管理アルゴリズムを容易に組み込むことが出来る。

(3) プログラムプロテクションの強化

現在の小型実時間 OS はプログラム保護機構が弱いので、プログラム障害等によりシステム全体がダウンすることが多い。また、組み込みシステムのプログラムは、リソースの管理や制御が必要で、従来はカーネルモードで実行させる必要があるので、プログラム開発が非常に困難であった。リソース管理プログラムもユーザタスクと同様にプロテクトされた環境で行える様にしたい。

(4) 多重処理

一般に制御システムは並行動作する多数のリソースやサービスを制御する必要があるので、効率的で融通性の高いマルチスレッド機能が望まれる。

(5) 分散処理機能

ネットワークを介して多数のリソースやサービスを協調動作させることが要求される。また、分散処理形態も Client-Server 型分散処理だけでなく、各 Entity が対等の立場で通信し合う Peer-to-Peer 型分散処理が今後は重要になる。また、メッセージのやりとりも同期型の他に非同期型通信が要求される。さらに、使い易いサービスの登録や検索機能が要求される。

(6) プロードバンド時代のサービスサーバとして、実時間ストリームデータを扱うためのサービス品質保証のための基本機構も OS が提供することが望まれる。

3 本研究のアプローチ

このような要求に応えるために、我々は以下のようなアプローチを採っている。

3.1 マイクロカーネル型 OS

OS 機能は大別して、メモリ管理・スレッド制御等のプリミティブ機能とファイルシステムやネットワークシステムなどの OS サービス機能とに分類できる。代表的な OS 構造には、プリミティブ機能から OS サービス機能までを一体化したモノリシック型 OS と、分離して後者を別タスク化したマイクロカーネル型 OS とがある。効率的には、高度のハッカーが適切に実装するならば前者が優れるが、組み

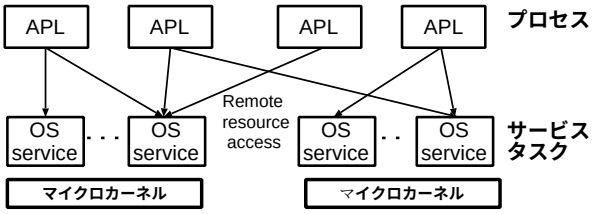

図 3.1: マルチサーバ構成

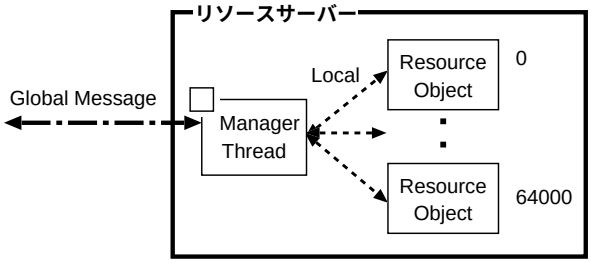

図 3.2: リソースサーバ

込みシステムの場合は OS サービスに相当する機能の大規模開発が必要であること、OS サービス機能もプロテクト環境で実装並びに実行させたいこと、優れたマイクロカーネルならば総合オーバヘッドは数%におさまり、むしろシステム全体の最適化によりトータル性能は向上が期待できる[1]。マイクロカーネルの方が、ユーザ独自の OS 機能を追加しやすく、またユーザ独自のスケジュールやメモリ管理のアルゴリズムを追加しやすい。

3.2 マルチサーバ OS

既存のマイクロカーネルを使った OS でも、リソース管理機能 (ファイルサービス、NW サービス等のいわゆる OS サービス機能) は単一のタスクとしているものが多い (*Cf.* L4Linux[3], MachLinux, etc.) が、制御システムの場合には多種多様なリソースを制御するため、個々の機能単位に独立タスクにすることもできることが望まれる (図 3.1)。

3.3 分散処理の OS サポート

融通性に富む分散処理、リモートリソースもローカルリソースと同様に扱えること等を実現するために、OS として以下のようなサポートを行なう。

(1) Peer-to-peer 分散処理のための機構のサポート

制御システムでは, Client-Server 型処理の他に、分散オブジェクトが対等の立場で非同期にメッセージを交信しあう分散処理 (ここでは Peer-to-peer 型処理と呼ぶ) が重要である。このために、スレッドはグローバル ID も持てる様にし、このグローバル ID 使って目的オブジェクトに非同期メッセージを送れるようにする。また、非同期メッセージに対して、非同期に返答を返せる機構 (Future メッセージ) も用意する (Remote Procedure Call ではなくメッセージを明示的に用いる)。

(2) リモートリソースの操作

非常に多数 (10^4のオーダ) のリソースをローカル・リモートを問わずに効率的かつ簡単に操作できるようにするために、リソースサーバは「要求メッセージを受けて目的リソースを制御する manager スレッド (能動オブジェクト)」と「個々のリソースを表す受動オブジェクト」からなる構成とする (図 3.2)。前者はグローバル ID を持ちグローバルにメッセージ交信が出来、後者は前者に従属して動作する。ユーザは、ネームサーバ、サービスブローカ若しくはグローバル名前空間を使って Manager Thread のグローバル ID を得ることにより、自在にリソースを操作できるようになる (*Cf.* File の NFS マウント)。

(3) サービスの登録・公開・検索

(4) 冗長性の確保のためのオブジェクトの複製と、メッセージのマルチキャスト

(5) 実装

これらの分散処理サポート機能は、基本的にはサービスサーバとして実装する予定であるが、効率の評価によってははマイクロカーネル内部への機能追加も考える。また、ユーザプロセスレベルでの Cache は、ユーザライブラリの中での提供も考える。

3.4 QoS 制御とプロトコルスタック

動的なプロトコルスタックの組込みによる柔軟なプロトコル処理を意図して、X-kernel[4] 等を参考に

[1]そのプロジェクトに Linus Torvalds 並のハッカーが永続して居るならば、モノリシック構造を採用するのは良いアイデアである

検討を進めている。

4 L4 マイクロカーネル

マイクロカーネルが提供すべき機能は、①論理空間の管理、②スレッド制御、③スレッド間通信、④割り込み対処である。独自開発も含めて目的に合うマイクロカーネルの検討をして、ドイツ GMD で設計された L4 マイクロカーネル [8] の利用が効果的であると判断した。L4 は、既存のマイクロカーネル (Mach[1], V[2], Chorus[11] 等) の技術蓄積が活かされており、必要ならば機能追加・変更をすることで、我々の目的を実現できる。L4 は、以下の特色を有する。

(1) タスク (論理空間) の管理

L4 は個々のタスクの論理空間を管理するが、物理ページとのマッピングを行う pager はユーザが独自に組み込むことが可能である。

(2) マルチスレッド機能

個々のタスクの中で最大 64 個のスレッドが使える。スレッドの仕組みは洗練されており、実時間システム向きの効率を提供する。

(3) メッセージ通信機能 (スレッド間通信機能)

マイクロカーネルの性能を一番左右するのが、スレッド間通信の効率であり、L4 はこの点で非常に優れている。L4 のスレッド間通信機能は、以下の特色を持っている。

(a) メッセージの宛て先はスレッドであり、ThreadID により指定される (*Cf.* Mach ではメッセージボックス宛)

(b) メッセージは同期型であり、送り元は宛て先が受理するまでブロックされる (*Cf.* Mach は非同期型)。

(c) 宛て先スレッドが同一タスク内でも、別タスクでも、同様にメッセージを送れる。

(d) 非常に効率的なパラメータ転送

- 32 bits × 3 個 (Ix86 の場合) のパラメータはレジスタのみで転送されるので、約 100 マシンサイクルで転送され [7]、手続き呼び出しよりも少し重いだけである (RISC や IA-64[6] プロセッサは多数のレジスタを持つので、更に多数のパラメータをレジスタ経由で転送できる)。
- それ以上の多数パラメータの場合はメモリコピーを伴うが、十分に効率的である。
- 最大 512K ワード (2 メガバイト) までのバッファー間コピーも行える。scatter and gather も可能。
- ページ単位のデータ転送は、ページマッピングだけで非常に効率的に行える。

(4) 割り込みの扱い

割り込みハンドラをカーネル内でなくタスク内に置くことが出来る。割り込みハンドラはスレッドとして実装し、そのスレッド ID を L4 に登録する。割り込みが生ずると、L4 は割り込みメッセージを割り込みスレッドハンドラに送り、そこで割り込みが処理されることになる。

なお、L4 の実装は数種類あるが、昨年 10 月にフリーソフトとして公開された C++で実装した Ix86 用カーネルは [9]、性能と簡明さがよくバランスした実装である。

5 マルチサーバ OS 構成

リソース管理機能のプログラム保護とモジュラリティーを強化するために、ファイルサービス、ネットワークサービスといったサービス単位毎に独立タスクとして実装する。この方式に関わる検討内容等を以下に説明する。

(1) プログラム保護と効率

個々のサービスが個別タスクとして走行するため、プログラム保護は頑強である。一方、TLB の入れ替え等タスク切り替えに付随するオーバーヘッドが生じるために実装によっては効率が損なわれる。そこで、現在以下の対策を検討している

Small space 機能の活用 L4 では、TLB の flush をせずにタスク切り替えを行なうためのアドレス空間が実現されている [8]。この機能を利用することにより、タスク切り替えのオーバーヘッドが削減され、機能を

複数のサーバに分割したオーバヘッドを軽減することが出来る。

ReadOnly 共用空間の利用 個々のサーバは一般にユーザプロセスの情報にアクセスする必要がある。プロセス情報を管理するタスクを設けてIPC経由でプロセス情報を逐一取得する方法も考えられるが、複数のサーバが効率よくプロセス情報にアクセスするために、各サーバが直接アクセス出来るようReadOnlyでマップされた共用空間にプロセス情報を配置することを検討している。

(2) 実行モード

サービスタスクはユーザモードで稼働させたいが、割り込みハンドラの作り方によってはカーネルモードが必要となる。その場合、割り込みハンドラをカーネル空間とユーザ空間のどちらに配置するかという選択肢がある。一般にハンドラは割り込みモードで実行されるため、何れの場合も処理に問題があるとシステム全体が倒壊することに注意が必要である。

カーネル空間に配置 適応型のOSという概念を維持するためにハンドラをカーネルにDynamic loadする機構をつけて (*Cf.* Spine)、サーバ本体はユーザモードとする (図 5.1左)。ハンドラがカーネル空間のデータを破壊出来ないように安全なloading機構が必要になる。ハンドラはサーバ本体に割り込みがあったことをIPCで通知する必要がある。

ユーザ空間に配置 ハンドラは適切にアドレス空間を配置すればカーネルから直接手続き呼び出しの形式で起動することも可能であるが、ここではカーネルに割り込みメッセージの送り先としてハンドラを登録しておき、割り込みをIPCメッセージとして受信する (図5.1右)。ハンドラがカーネル空間のデータを直接倒壊出来ない点での安全性が確保出来るが、カーネルがハンドラをスケジュールすることで割り込みモードでハンドラに制御が渡る点ではカーネル空間に配置した場合と同様であり、ハンドラが処理を終了して次のIPC待ち (割り込み待ち) システムコールを発行する迄は割り込みモードが継続しているため依然処理自体の安全性の確保が要求される。

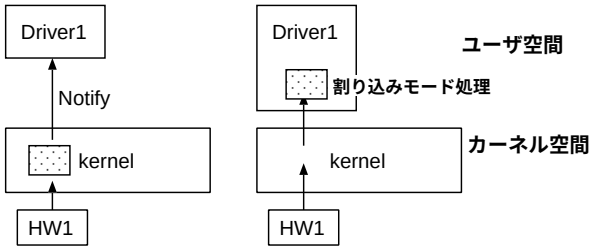

図 5.1: IRQ ハンドラの配置

なお、何れの場合もLinuxにおけるbottom halfに相当する処理はユーザモードで実行可能である。

(3) I/O 空間のアクセス権

ドライバプログラムは、一般にI/O制御空間 (あるいはmapped IO空間) にアクセスする必要がある。従って、サービスタスクにはI/O空間のアクセス権を与える必要がある。

Ix86の場合、アクセス制御に

- TSS の I/O map base address が指す bit map による制御
- EFLAGS register の I/O privilege field による制御

の2通りの手段が提供されている [5]。

(4) IRQ sharing

IRQ番号は一般には複数のデバイスで共有されるため、デバイスへのアクセスを担う複数のサービスタスク間で共有する必要がある。共有する方法として、割り込みメッセージを

(a) カーネルがマルチキャストする (5.2左)

(b) サービスタスクが転送する (5.2右)

という選択肢が考えられる。何れの場合もハードウェアに割り込み確認を出すタイミングに注意する必要がある。

現在、Minix[13] の HDD(IDE) ドライバと Minix ファイルシステムを、ファイルサービスタスクとして L4 の上に移植を進めている。

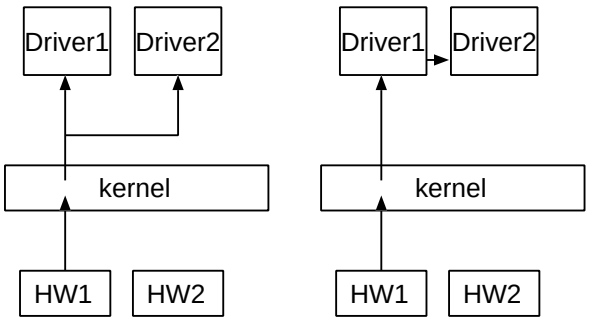

図 5.2: IRQ の共有

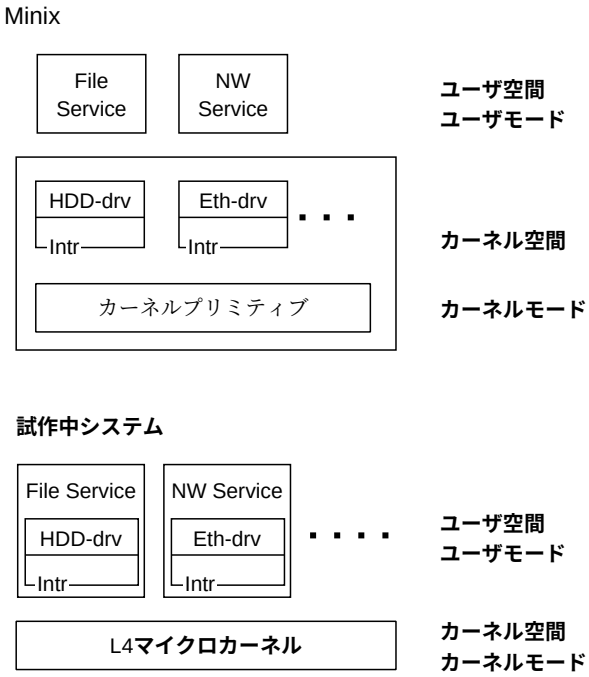

図 5.3: 試作中のシステム

6 分散処理機能

汎用の分散処理ツール (CORBA [10], Java-RMI [12] 等) では、リモートオブジェクトの Proxy/Stub をローカルに用意して、クライアントが Proxy/Stub を呼ぶと Remote Procedure Call により宛先ノードの Stub スレッド経由で目的オブジェクトが実行される。これに対し、制御システムの分散処理が要求する性能・信頼性・融通性を OS レベルで実現するために、リモートスレッドと自在に直接通信できる事が望まれる。このために、L4 マイクロカーネルの外部もしくは内部 (性能上) に以下の機能を追加する。

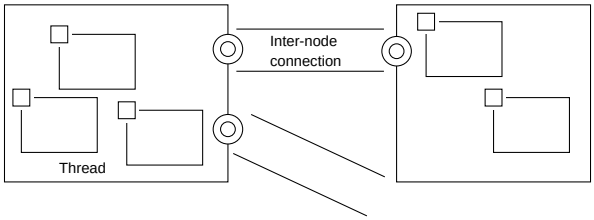

図 6.1: ノード間のコネクション

(1) グローバルスレッド ID (GThID)

L4 マイクロカーネルのスレッド ID はローカル ID であり、基本的には<TaskID, LocalID>からなる。分散処理環境では、これに NodeID を追加した GThID: <NodeID, <TaskID, LocalID>> をメッセージの宛て先として使えるようにする。

(2) ノード間のコネクション

通信でのメッセージ欠落を避けるために、Connection を張り通信制御を行う。Connection はスレッド間でなくノード間に張ることにより、静的・動的に効率を上げる。各ノードは、分散処理の開始時に相手ノードとコネクションを張る。以降は、NodeID により相手ノードが定まる。なお、現在の試作ではノード間通信には (Streeam 転送を意図した) TCP を用いているが、本来はメッセージ通信に適合し、かつ信頼性の確保された (メッセージが確実に送り届けられる) プロトコルが望ましい。

(3) 非同期メッセージ転送と非同期返答の機構

(4) ネームサーバ・トレーダ・リモートマウント

リモートアクセスのためのサービスやリソースの登録と検索の機構は、普通のサービスタスクとして簡単に実装できる。

本 OS のサービスサーバは、分散サーバとして実装するので、リモートリソースもローカルリソースと同様にアクセスできる。なお、応用プログラムとリンクするライブラリにて、(ファイル入出力の際のバッファキャッシュ等を対象とした) ローカル Cache 機能を提供することも検討している。

7 おわりに

実時間制御システムに共通に必要な OS の諸性質について検討を行ない、拡張性、高信頼性、高度な分散処理をサポートする事が必要であることを明らかにした上で、最新のマイクロカーネル技術とマルチサーバ方式によりそのような OS が構成出来ることを示した。

現在 L4 マイクロカーネル上にマルチサーバ OS である Minix の移植を進めている。移植の仮定で割り込み処理と保護機能の問題、論理空間切り替えの効率の問題等が表面化しており、様々な解決法を検討した。更に、分散処理のために必要な枠組についても検討を行なった。

謝辞

本稿をまとめるにあたって広く御討論頂いた国際基督教大学・中村明教授、国立情報学研究所・橋爪宏達教授、計宇生助教授、井手一郎助手に深く感謝致します。

参考文献

[1] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for UNIX development. *Proceedings of Summer 1986 USENIX Conference*, July 1986.

[2] David R. Cheriton. The V Distributed System. *Communications of the ACM*, Vol. 31, No. 3, March 1988.

[3] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The performance of μ-Kernel-based systems. In *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)*, Vol. 31,5 of *Operating Systems Review*, pp. 66–77, New York, October 5–8 1997. ACM Press.

[4] Norman C. Hutchinson and Larry L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, Vol. 17, No. 1, pp. 64–76, January 1991.

[5] Intel. *Intel Architecture Software Developer's Manual*, 1999. ftp://download.intel.com/design/PentiumII/manuals/24319102.PDF.

[6] Intel Corporation. *Intel IA-64 Architecture Software Developer's Manual*. Intel Corporation, Santa Clara, CA, USA, January 2000.

[7] Jochen Liedtke. Improving IPC by kernel design. In *Proceedings of the 14th Symposium on Operating Systems Principles (14th SOSP'93), Operating Systems Review*, pp. 175–188, Asheville, NC, December 1993. Published as Proceedings of the 14th Symposium on Operating Systems Principles (14th SOSP'93), Operating Systems Review, volume 27, number 5.

[8] Jochen Liedtke. On μ-Kernel Construction. *Operating Systems Review*, Vol. 29, No. 5, pp. 237–250, December 1995.

[9] Johen Liedtke. *L4 Nucleus Version X Reference Manual x86*. Universität Karlsruhe, September 1999. http://l4ka.org/projects/hazelnut/docs.asp.

[10] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, July 1996.

[11] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Lé oñard, and W. Neuhauser. CHORUS distributed operating system. *Computing Systems*, Vol. 1, No. 4, pp. 305–370, Fall 1988.

[12] Sun. Java remote method invocation specification. Technical report, Sun Microsystems, 1997. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

[13] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems: Design and Implementation*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, second edition, 1997. Includes CD-ROM.