システムソフトウェアと
オペレーティング・システム　71－7
マルチメディア通信と分散処理　73－7
（1995．11．30）

単一仮想記憶空間を提供するリアルタイムOSの設計

河野通宗　　前沢敏行　　安西祐一郎

慶應義塾大学 計算機科学専攻

E-mail: {kohno,toshi,anzai}@aa.cs.keio.ac.jp

リアルタイムオペレーティングシステムに単一アドレス空間システムを適用することを試みる。単一アドレス空間システムは、分散システムやオブジェクト指向の研究において注目されつつあるメモリ管理法である。タスクごとに別のアドレス空間を割り当てるのが現在主流のメモリ管理法では、タスク間のメモリ共有の機構が複雑になっていた。これに対して単一アドレス空間は、ハードウェアによる保護のもとで、すべてのタスクが1つのアドレス空間で動作する。本稿ではこのメモリ管理法をリアルタイムOSに適用した場合の利点、および問題点について述べる。さらに、我々の研究室で開発中のロボット用OS μ-PULSERへの実装方針を示す。

Design of Real-Time Operating System with Single Address Space Architecture

Michimune Kohno　Toshiyuki Maezawa　Yuichiro Anzai

Department of Computer Science, Keio University

This paper describes how to apply single address space on real time operating systems. A Single address space is the memory management technique different from recent multi address-space system. With this technique, we are capable of reading data on other tasks without any overhead. Nevertheless, data in my own task is protected from any other tasks. We discuss about pros and cons of real-time operating system with single address space. We also describe some difficulties with current major processors. Finally, we show how to implement this technique into our Robot operating system, μ-PULSER.

1 はじめに

ロボットを動作させるタスクが要求するサービスの大部分は、モータの駆動やセンサからの情報の取得などのI/O処理である。これらのI/O処理はすべてマイクロカーネル上で実行しているサーバタスクが行なっている。サーバは各々独立したアドレス空間上でサービスを行なう。そのために、ユーザタスクからのリクエストに伴うコンテキストスイッチによるオーバヘッドが、システム全体のパフォーマンスに大きく影響している。

我々が自律移動ロボットのために開発したオペレーティングシステムμ-PULSER[1]はリアルタイム性を持つユーザレベルスレッド機構を提供している。これによってμ-PULSERはシステム全体を通してスレッドの時間制約を保証し、かつタスク内における高速なコンテキストスイッチを実現している。しかしながら、別タスクのスレッドとの通信ではカーネルが介在しなければならないため、ユーザレベルスレッドの利点を活かせなかった。

そこで本稿では、上記の問題点を解決するため、近年盛んになりつつある単一アドレス空間によるタスク・スレッド管理機構をμ-PULSERに実装する。単一仮想記憶空間(以後Single Address Space: SASと呼ぶ)は64ビットプロセッサの登場に伴って提唱されはじめ、いくつかの実装も行なわれた[2, 3, 4]。しかしSASをリアルタイムシステムに適用したという例は見られない。SASの問題点はタスクのメモリ保護が難しいという点であるが、これを解決する手法もいくつか提案されており、μ-PULSERの抱える問題点の解決に大きく寄与するものと考える。本稿では、SASをどのようにμ-PULSER上に実装するかについて述べる。

2 現状の問題点

本節では、まず、我々の開発しているロボットで動作するアプリケーションの、各サーバへのリクエストの回数を示す。その上で、マイクロカーネルアーキテクチャにおける通信オーバヘッドの問題について述べる。

2.1 ユーザタスクのサービス要求回数の測定

現状のμ-PULSERにおける問題点を明らかにするために、ロボットを動作させる2つのアプリケーションを用いて評価を行なった。具体的は、ユーザタスク(アプリケーション)による、μ-PULSER上で実行している各種サーバへの、サービス要求回数の割合を測定した。

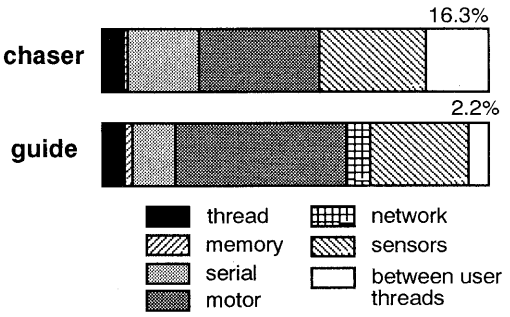

図1: アプリケーションのサービス要求回数の割合

まず評価したアプリケーションに関して簡単に説明する。guideはロボットが道案内をするタスクで、chaserはロボットが人の発話する方向を認識し、追随して動くタスクである[5]。次に測定を行なう環境について説明する。ロボットは3つのモジュールから構成されている。各モジュールにはプロセッサとメモリが搭載されており、これらのモジュールがVMEバスで結合されている。アーキテクチャの分類は、分散共有メモリ型のマルチプロセッサアーキテクチャである[6]。プロセッサは、1つのモジュールがMC68030 25MHzであり、残りの2つはTMP68301 16MHzである。各モジュールにはそれぞれ固有のデバイスが接続されており、個々のモジュール上でオペレーティングシステムが独立して動作している。別のモジュール上のサーバへのリクエストは、共有メモリ内に割り当てたポートに書き込むことにより行なう。MC68030を搭載したモジュール上で評価対象のアプリケーションを実行させ、そのタスクが要求したサービスの回数をカウントした。図1は、この結果を百分率で表したグラフである。

図中の一番右の白い部分は、アプリケーション内のスレッド間で同期を行なっている割合を示している。図を見ると明らかなように、アプリケーション内、すなわちあるタスク内の同期よりも、別タスク内に存在するサーバへのサービス要求の方が圧倒的に多いことがわかった。

表 1: コンテキストスイッチのコストの比較

カーネルスレッド間	76.9 μs
ユーザレベルスレッド間	25.0 μs

2.2 通信オーバヘッド

μ-PULSER はカーネルレベルスレッド[1]に加えて、ユーザレベルスレッドによる高速なコンテキストスイッチ、および同期機構を実装している。表 1 に、カーネルレベルスレッド同士、およびユーザレベルスレッド同士での、コンテキストスイッチに要する時間を示す (同一アドレス空間内)。同一タスク内のユーザレベルスレッド間の通信は、共有変数にデータを書き込んでコンテキストを移すだけでよいので、そのコストは、メッセージによるサーバとの通信に比べてかなり小さい。

各種サーバはそれぞれ独立したアドレス空間内で実行しているため (タスク管理スレッドとメモリ管理スレッドを除く)、ユーザタスクがサービスを要求する際にはカーネルが介在する必要がある。したがって、通信に必要なコストはカーネルレベルスレッド間通信のコストと等しくなる。

2.3 問題点のまとめ

ロボットを動作させる 2 つのアプリケーションについて、スレッド間通信の回数を測定した。その結果、各種サーバとの通信回数が、同一タスク内のスレッドとの通信に比べてはるかに多いことがわかった。そのために高速なユーザレベルスレッド間の通信はほとんど行なわれず、ユーザレベルスレッドの利点が活かされていないことがわかった。本論ではこの問題点をふまえ、μ-PULSER の持つリアルタイム性を失うことなく、ユーザレベルスレッドの利点を活かせるようにオペレーティングシステムを改善することを目的とする。

3 単一仮想アドレス空間

単一仮想アドレス空間は、アドレス空間が 64 ビットに拡張された際の、アドレス空間の有効な使用法について検討された結果、議論され始めたメモリ管理方法である [7]。比較のため、まず既存のオペレーティングシステムのメモリ管理法について簡単に述べる。その後、今までに提案されている単一仮想アドレス管理方法に関して、主にメモリ保護の観点から述べる。

3.1 現在の主な OS におけるメモリ管理

16 ビットや 32 ビットプロセッサをターゲットとしたオペレーティングシステムでは、各タスクに個別の仮想アドレス空間を与えるメモリ管理方法が広く一般的に用いられてきた。タスクに割り当てられていないメモリ空間に対するメモリアクセスはハードウェアによって検出され、ページフォルトハンドラが必要な処理を行なった後にタスクを再開させるか、終了させる。仮想アドレスへの変換、および未割り当てページへのアクセスの検出は MMU が行なう。MMU によるメモリ管理はページ単位で行なわれる。ページサイズはアーキテクチャによってまちまちであるが、大抵 4KB から 8KB までである。各ページに対応するページテーブルエントリには読み込み、書き込み (アーキテクチャによっては実行) 可能ビットがあり、許可されていないアクセスに対してはトラップを発生させる。既存のオペレーティングシステムは、これらの機構を用いて Copy on Write や Map on Reference などの遅延評価を実装し、システムの処理性能を向上させてきた [8]。

Mach に代表されるマイクロカーネルアーキテクチャのオペレーティングシステムでは、ユーザレベルで動作するサーバタスクがアプリケーションにサービスを提供する。アプリケーションがサーバにリクエストを送る際にはメッセージか共有メモリによる IPC が用いられる。したがって IPC の最適化はシステム全体の速度向上に寄与することを容易に想像させる。しかし Chen らは [9] の中で、Mach における IPC のオーバヘッドはシステム全体のオーバヘッドに比べて小さく、IPC のみを主眼とする最適化はそれほどパフォーマンスに影響を与えないと報告している。さらに Chen らは、より効率的にオペレーティングシステムを改善するためには、IPC だけでなく、ページマップのストラテジなどメモリシステム全体について考慮すべきだと述べている。

3.2 単一仮想アドレス空間のメモリ保護

単一仮想アドレス空間では、全てのタスクが 1 つの仮想アドレス空間を共有する。スレッドは自分の属する保護ドメイン (protection domain) 内で実行される。保護ドメイン内のデータは、他の保護ドメイン内のスレッドによる、不正なデータ書き込みか

[1] カーネルレベルで実装されているスレッドをカーネルレベルスレッドと呼ぶことにする。

ら守られなければならない。この保護メカニズムに関して、いくつかの手法が提案されている。

Koldinger らは [7] で、domain-page モデルと page-group モデルの 2 つを提案している。domain-page モデルは、(domain, page) の組み合わせごとにアクセス権を明示的に指定する。現行のアーキテクチャで使われている TLB からドメインタグを取り除き、PLB(protection lookaside buffer) の中に格納しようというものである。一方 domain-group モデルは論理的にグルーピングされたページ群で保護ドメインを定義する。アクセス権はグループ内の各ページが持つ。このモデルは、ヒューレットパッカードの PA-RISC で使われている。

Chase らは、オブジェクト指向的な観点から、単一アドレス空間の有効性に関して述べている [10]。この論文では、連続した仮想アドレス空間である仮想セグメントを、保護ドメインを構成する基本単位にしている。ドメインは通常、少数の仮想セグメントに対するパーミッションを持つ (read, write, execute)。ドメイン間の通信には、portal をケーパビリティに用いる。portal はあるドメインへのエントリポイントであり、portal の値を知っているスレッドだけが、そのドメインへ制御を移すことを許可される。また同じ論文の中で、ユーザレベルのスレッド機構と同期メカニズムが、オブジェクトの共有を単一アドレス空間システムで実現する際に本質的であると述べている。

3.3 リアルタイムシステムと単一アドレス空間

前節で述べたように、オブジェクト指向の観点、あるいは分散システムの観点から単一アドレス空間システムに関して述べた文献はいくつかあるが、リアルタイムシステムに適用した場合の利点、問題点に関して議論したものはみられなかった。そこで、単一アドレス空間システムをリアルタイムオペレーティングシステムに適用した場合について、我々の実装対象である自律移動ロボットを考慮にいれて議論する。なお本稿では、単一アドレス空間の、分散システムへの拡張は考慮しない。

まず、現在商用で用いられているリアルタイム OS について説明する。商用リアルタイム OS は、複数のタスク空間を独立させているものと、アドレス変換を一切行なわないものとに分かれる。複数タスク空間システムでは、通常の OS と同様、共有メモリを定義してタスク間通信を行なう。アドレス変換を行なわないシステムではタスク間の区別はないので、実質的な保護機構は存在しない。

リアルタイムシステムでは、スレッドの時間制約を守ることが要求される。リアルタイムスケジューラは、システムで実行可能なスレッド全部の中からもっとも優先度の高いスレッドにプロセッサを割り当てる必要がある。しかし、ユーザレベルで実装されたスレッドスケジューラは、そのスケジューラに属するスレッド以外の優先度情報を取得できないので、リアルタイム性を保証するのが難しい。追川らはこの問題の解決手法を [11] で示している。追川らの提案は、全てのアドレス空間で共有されるスケジューラを設け、その中でユーザレベルスレッドの情報を管理するものであった。

次に、単一のアドレス空間でリアルタイムユーザレベルスレッドを提供することを考える。適切なハードウェアのサポートが得られるならば、今までのシステムで用いられた以下の手法に関して、簡略化あるいはオーバヘッドの削減が可能と考える。

- **スケジューリング:** ユーザレベルスケジューラは、特殊なメモリマッピングストラテジを用いることなく、全ての実行可能スレッドの優先度を取得できる
- **メッセージ:** サーバへのメッセージ転送が、ユーザレベルで実行可能
- **デバイス駆動:** メモリマップされたデバイスを、ユーザレベルのみで安全に操作できる

もっとも理想的なのは、ドメイン全体を通してすべてのスレッドがユーザレベルで管理されることである。ロボットのようなハードリアルタイムシステムにおいては、柔軟なスケジューリングポリシを選択できることも重要であるが、それよりむしろデッドラインミスの確率をできる限り低くすることが要求される。この要求を満たすためにはシステムの予測可能性を高めることが不可欠である。予測可能性を高めるということは、システムの予測不可能なオーバヘッドを低減させることと等しい[2]。もし 1 つのユーザレベルスケジューラのみで全体のスケジューリングが可能であり、かつ高速にコンテキストスイッチできるのであれば、システムの挙動の安定性が大きく向上すると考える。この安定性とドメイン保護の両立が、単一仮想アドレス空間ならば可能であると確信する。

[2] 優先度逆転問題の対策は実装を完了している。

単一アドレス空間の問題点は、エミュレーションによる実装が困難なことである。現在の主なプロセッサは複数アドレス空間でのタスク実行を前提としている。そのため、保護ドメインから保護属性への変換 (第 3.2 節の PLB) 機構が存在しない。この保護機構のエミュレーションは 2 通り考えられる。一つはコンテキスト ID ごとにすべてのページテーブルを作成し、すべてのページテーブルが同じアドレス変換を行なう方法である。そしてページ属性だけを書き換えることによって保護機構を実現する。しかしこれではページテーブルに使う領域が大きい上に、スーパバイザモードでコンテキスト ID を切り替える必要があり、現行のシステムと大差なくなってしまう。もう一つの方法は、ページフォルトを用いたエミュレーションである。これは細粒度の保護が実現できるので、より実際のシステムの実装に近づけることができる。しかし、頻繁にページフォルトが発生して処理速度が著しく低下する可能性が高い。特にリアルタイムシステムにおいては、散発的なトラップはスケジューラビリティを低下させる危険を含む。実用に耐えられるように設計するならば、ケーパビリティを用いた粗い粒度の保護機構を実装する必要があると考える。

またエミュレーションを 32 ビットプロセッサで行なう場合、仮想アドレスの断片化が問題になる。64 ビットよりはるかに小さいアドレス空間である 32 ビットアドレスは、枯渇する可能性がある。これはセグメンテーションによる物理メモリの外部断片化と同様の問題である。この対策としてはフラグメンテーションの圧縮が挙げられるが、計算時間の予測可能性が低いのでリアルタイムシステムに適用するのは難しい。

4 設計

前節において、現在主流である複数アドレス空間システムと、単一アドレス空間システムのメモリ管理について概観した。またリアルタイムシステムに単一アドレス空間を採用した際の利点、欠点について述べた。本節ではこれらをふまえ、単一アドレス空間システムの μ-PULSER の設計方針について説明する。

4.1 設計対象について

ターゲットアーキテクチャは μ-SPARC 40MHz、メモリ 32MB のマシンである。しかし、SPARC アーキテクチャは単一アドレス空間を実装するには適しておらず、エミュレーションする必要がある。その理由は 2 つある。1 つは第 3.3 節でも述べた通り、SPARC アーキテクチャは複数アドレス空間での使用が前提であり、ページテーブルをドメイン (アドレス空間) 毎に持つ必要があるためである。コンテキスト ID をドメイン ID に用いるならば、同じアドレス変換を行なう無駄なページテーブルをメモリ中に持たなければならない (保護属性はドメイン毎に異なるので、同じテーブルを参照できない)。もう 1 つの理由もページテーブルに関する問題である。ページテーブルをコンテキストごとに持っているため、ある仮想ページの保護属性を変更した場合、ほかのすべてのコンテキストに属する PTE(Page Table Entry) の保護属性を変更しなければならない。このオーバヘッドは計算可能であるが、システム全体のオーバヘッドに対して占める割合が無視できないものになることが考えられる。

4.2 タスク・スレッド

今までのシステムでは、タスクは資源の割り当て単位であり、かつアドレス空間の割り当て単位であった。この定義をドメインの割り当て単位に変更する。保護の粒度はページサイズと等しくする。コード領域には読み込み、実行許可属性を全てのドメインに対して与え、リロケータブルコードを格納する。データ領域は、自分の属するドメインのみ読み込み・書き込み許可フラグをセットする。データ領域とは別にポート用の領域を設け、すべてのドメインに対して公開する。このポートのアドレスはポートマネージャが管理する。コード・データ領域共に、全てのドメインで等しい仮想アドレスにマップする。またスレッド情報構造体を全てのドメインから参照可能にし、どのドメインからでも、スケジューラがキューの更新を行なえるようにする[3]。ユーザレベルスケジューラはリアルタイム性を保証することを重視して OS が提供する。これにより、事実上スケジューラの実体を 1 つに統合する。

4.3 通信

他ドメインの関数を直接呼び出せるのが理想であるが、保護の粒度が粗いため、従来と同じくポートを用いたメールボックス型のメッセージ通信を採用する。コンテキストスイッチは、ドメイン ID の変更 (コンテキストレジスタ書き込み)、および割

[3] この部分に関する保護の問題は今後の課題である。

り込みレベル設定以外をできる限りユーザレベルで行なう。あるスレッドがCPUを解放したら、そのスレッドの属するドメイン内でユーザレベルスケジューラがスケジューリングし、その結果を受けて、次にプロセッサを割り当てるスレッドに直接ディスパッチする。外部割り込みが発生したときは、割り込みを受け取ったドメインで低レベルの処理が行なわれる。μ-PULSERはハードウェア割り込みとソフトウェア割り込みを統合したコンテキスト切り替え機構を実装しており[1]、割り込み処理後に特定のスレッドの起動が必要な場合、スケジューラが起動されて再スケジューリングされる。

複数アドレス空間に対して優位な点は、コンテキストの切り替え時にキャッシュエントリをフラッシュする必要がないことである。複数のアドレス空間を持つ場合、ダイレクトマッピング方式のキャッシュエントリであっても衝突が発生する。単一アドレス空間では、キャッシュエントリの衝突による一貫性の喪失がないため、無駄なフラッシュを行なう必要がない。通信の際にコンテキストを頻繁に切り替える必要のあるシステムにおいて、このことは処理速度に大きく影響すると考える。

5 まとめ

リアルタイムオペレーティングシステムへの、単一アドレス空間システムの適用に関して議論した。自律移動ロボットにおけるスレッド間通信は、ユーザタスク内よりもタスク外のサーバとの通信が多いことを示した。そしてユーザレベルスレッドの利点を活かす方法として単一アドレス空間によるアプローチを提案し、具体的な設計方針について述べた。現在、単一アドレス空間のメモリ管理ルーチンをμ-PULSERに実装中である。今後コンテキスト切り替え時間、CPU利用率、非周期的な負荷に対する耐性などを評価する予定である。

参考文献

[1] 矢向高弘, 菅原智義, 安西祐一郎. μ-PULSER: パーソナルロボットを構築するためのオペレーティングシステム. 電子情報通信学会論文誌 D-I, Vol. J77-D-I, No. 2, pp. 207–214, February 1994.

[2] Timothy Roscoe. Linkage in the Nemesis single address space. *ACM Operating Systems Review*, Vol. 28, No. 4, pp. 48–55, October 1994.

[3] Jochen Liedtke. Address space sparsity and fine granularity. *ACM Operating Systems Review*, Vol. 29, No. 1, pp. 87–90, January 1995.

[4] 岡本利夫, 津田悦幸, 福本淳, 寺本圭一. 次世代アーキテクチャ向けオペレーティングシステムの開発. 情報処理学会システムソフトウェアとオペレーティングシステム研究会報告, 94-OS-66, pp. 17–23, 9 1995.

[5] Nobuyuki Yamasaki and Yuichiro Anzai. *Active Interface* for Human-Robot Interaction. In *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. Vol.3, pp. 3103–3109, Nagoya, Japan, May 1995.

[6] 山崎信行, 安西祐一郎. パーソナルロボットのためのアーキテクチャの提案. 日本機械学会ロボティクス・メカトロニクス講演会論文集, pp. 51–56, 1992.

[7] Eric J. Koldinger, Jeffrey S. Chase, and Susan J. Eggers. Architectural support for single address space operating systems. In *Proceedings of 5th International Conference on Architectural Support for Programming Language and Operating Systems*, pp. 175–186. Vol. 26 of ACM SIGPLAN Notices, October 1992.

[8] R. Fitzgerald and R. F. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Transactions on Computer Systems*, Vol. 4, No. 2, pp. 147–177, May 1986.

[9] J. Bradley Chen and Brian N. Bershad. The impact of operating system structure on memory system performance. *ACM SIGOPS*, pp. 120–133, December 1993.

[10] Jeffley S. Chase, Henry M. Levy, Edward D. Lazowska, and Miche Baker-Harvey. Lightweight shared objects in a 64-bit operating system. In *Proceedings of 5th International Conference on Architectural Support for Programming Language and Operating Systems*, pp. 397–413. Vol. 25 of ACM SIGPLAN Notices, 1992.

[11] 追川修一, 徳田英幸. 外部スケジューラに基づくマイクロカーネルの構成. コンピュータソフトウェア, Vol. 11, No. 5, pp. 31–43, 1994.

[12] Takahiro Yakoh and Yuichiro Anzai. A new reactive operating system for human-robot interaction. *Advanced Robotics*, Vol. 8, No. 4, pp. 371–383, 1994.

[13] 徳田英幸. 分散リアルタイムOSの技術動向. コンピュータソフトウェア, Vol. 9, No. 3, pp. 182–192, May 1992.

[14] Yutaka Ishikawa, Hideyuki Tokuda, and Clifford W. Mercer. Object-oriented real-time language design: Constructs for timing constraints. In *Proceedings of 5th OOPSLA Conference*, Vol. 25, pp. 289–298. ACM SIGPLAN Notices, October 1990.