システムソフトウェアと
オペレーティング・システム 63－2
（１９９４．３．２２）

分散共有メモリを用いた実時間ページングの提案と実装

河野 通宗　飯田 浩二　矢向 高弘　安西 祐一郎

慶應義塾大学

リアルタイムOSで動作可能なページング機構を実装する。実メモリサイズに依存しないアドレス空間を提供するためにはページング機構を持つ仮想記憶が必要であるが、ページングは実行時間が予測可能でないために実装できなかった。そこで本研究では最悪実行時間を規定可能にすることにより、リアルタイムOSに実装可能なページング機構を提案する。そしてバス結合型分散共有メモリを用いて実装し、評価を行った。その結果、ページングの処理時間はバスの負荷や周期的負荷にさほど依存しなかった。しかし最大プロセッサ利用率はスケジューリングアルゴリズムに依存するため、スケジューラも含めた検討が必要であることが分かった。

Real-time paging using a distributed shared memory system

Michimune Kohno　Kouji Iida　Takahiro Yakoh　Yuichiro Anzai

Keio University

In order to provide a large address space which is not limited by physical memory size, an operating system must support virtual memory system with paging. Since the execution time of paging is not predictable, paging is not used on real-time systems. This paper proposes a paging technique that is available for real-time operating systems. We implemented and evaluated it on a bus-connected distributed shared memory system. We found that the process time of paging does not depend much on neither bus utilization nor periodic load. Since the breakdown utilization of processors depends on scheduling algorithms, more examination with a scheduler is needed.

1 はじめに

仮想記憶機構は、オペレーティングシステムにとって重要な機能の一つである。仮想記憶機構はプロセスのアドレス空間を保護するだけでなく、ページングを用いることによって実メモリ空間以上のメモリ空間を提供することが可能となる。しかしページング機構をリアルタイムOSに実装すると、ページングの実行時間が予測不可能であるため、リアルタイムスケジューリングができなくなる。例えばリアルタイム制約を持つプロセスがスワップアウトされていた場合にデッドラインミスを引き起こしたり、システムに致命的な影響を与える危険が生じる。ページングにはこのような欠点があるため、リアルタイムシステムに仮想記憶機構を実装する際にはページングを排除しなければならなかった。

本稿では、最悪実行時間の規定可能なページング機構を持つ仮想記憶の、リアルタイムオペレーティングシステムへの実装方法を提案する。またバス結合型分散共有メモリをページングデバイスとして用いて本機構を実装した。ページアウトをリアルタイム性のないタスクのアドレス空間に限定し、ページイン／アウトを別々のスレッドに分けてその処理が必要になる前に (積極的に) 行うことにより、リアルタイムタスクの実行を妨げないページングが実現できた。評価の結果、実行時間がバスの負荷や周期的負荷にあまり依存しないページングが可能であることが分かった。

2 リアルタイムOSへのページング実装方法の提案

ここではまずページングが既存のリアルタイムOSの仮想記憶機構に不適当である理由を示した後で、リアルタイムOSにページングを実装するための手法について述べ、さらにその際に予想される問題点について述べる。

2.1 ページングの問題点

リアルタイム性のないオペレーティングシステムでは、仮想記憶を実現する手法としてデマンドページングが広く用いられている。そして、そのページング先のデバイスとして多くのOSがハードディスクを用いている。しかし一方で、リアルタイムOSにページングが実装された例は見ない。それはページングがリアルタイムOSに不適当だからであり、以下ではその理由について述べた後で、ページングをリアルタイムOSに実装するために必要な条件を示す。

2.1.1 ページング処理時間の予測不可能性

既存のOSでページングデバイスにハードディスクが用いられてきたのは、ハードディスクが

- 1ビット当たりのコストが低く、かつ大容量な永続的メモリ
- ファイルシステムと統合した管理が可能

であるためである。しかし、ページングに要する処理時間を予測することは不可能である。Andersonら [1] や Ranganら [8] によって、ユーザの指定した転送レートを保証できるファイルシステムなどが研究されているが、これは continuous media データの連続転送を目的としたものであり、ページングに適用した例は見ない。その理由は、転送レートの保証のために大量のメモリを使うことは、少ない実メモリで広いアドレス空間を得るというページングの基本思想とは正反対であるためだと考えられる。現在の主流はデマンドページングであり、そのため現在ファイル管理に使われているディスク管理方式では処理時間の予測が不可能であり、リアルタイムスケジューラはページングをスケジューリングできない。

2.1.2 ページングによるデッドラインミス

リアルタイムOSでは、タスクの実行時間が予測可能であることが前提条件である [6, 10]。リアルタイムタスクのデッドラインはその周期や到着時刻から決定されるが、デマンドページングを用いているとタスクの処理時間にページングの処理時間が加わる。仮にページングの処理時間が予測可能であっても、タスクの処理時間が長くなったために、全てのタスクのデッドラインを満足するスケジューリングができなくなる可能性がある。

2.1.3 非周期的タスクのレスポンスタイムの悪化

ソフトデッドラインを持つ非周期的タスクではそのレスポンスタイムをできるだけ短くすることが要求される [9]。これはリアルタイム性を持たないタスクも同様であるが、このようなタスクの

ページがページアウトされていた場合、実行時間にページフォルト処理の時間が加わり、レスポンスタイムは悪化する。これは前節と同じくデマンドページングに起因する問題である。

2.1.4 ページング実装のための条件

以上をまとめると、ページングをリアルタイムシステムに実装するためには、

- 最悪アクセス時間が保証可能なページングデバイスを用いる
- デマンドページングなどの遅延評価を用いずにページングを実装する

必要があると考えられる。

2.2 実時間ページングの提案

本節では、前節で述べた条件を満たすページングの実装方法を提案する。この方法により、リアルタイムOSへのページングの実装が可能となる。

2.2.1 ページインアルゴリズム

ページが必要になってからページインするのではなく、メモリにある程度の空きができた時点で予めページインを行っておき、ページフォルトの発生を抑える。これを積極的なページングと呼ぶことにする。こうすることにより、2.1.3節で述べたようなレスポンスタイムの悪化を防ぐことができる。この積極的なページングのために空きメモリ容量S_{free}に対してある閾値S_{page_in}を設け、$S_{free} > S_{page_in}$となった時にシステムがページインを実行しておく。ページインの対象とするページの選択のアルゴリズムには、性能には影響するが本提案の有効性には直接影響しないので、本実装においては単純なFIFOアルゴリズムを用いることにする。

2.2.2 ページアウトアルゴリズム

深刻なメモリ不足が発生してからページアウトしていては、その間にデッドラインミスが起きるかも知れないし、レスポンスタイムも悪化する。これを避けるためページアウトの実行も、積極的に行う。ページアウトするページの選択にはLRUアルゴリズムを用いる。LRUは局所性に対する親和性がよいことからこれを採用した。ページアウトにもページインと同様に閾値S_{page_out}を設定し、$S_{free} < S_{page_out}$になった時にページアウトを実行するようにする。2つの閾値S_{page_out}とS_{page_in}

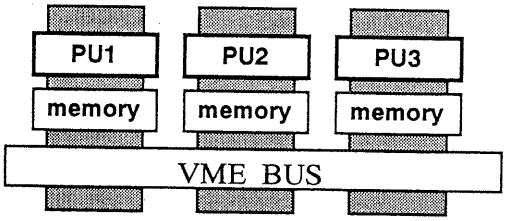

図1: バス結合された分散共有メモリ

の関係は$S_{page_out} < S_{page_in}$とし、差を適当にとる。この差は実メモリサイズから決定するが、この差に依存してスラッシングが起きる可能性があるため、慎重に決定する必要がある。

2.2.3 ページアウトを許すタスクの限定

2.1.2節で述べたデッドラインミスを回避するために、ページアウトされるタスクのページを、リアルタイム性を持たないタスクのページに限定する。リアルタイムタスクに関しては既存のリアルタイムOSと同様にページングは行わない。こうすると、リアルタイムタスクの要求するメモリ量の実メモリ全体に占める割合が非常に大きい場合には、ページング可能なページ数が少なくページングが有効に働かなくなる可能性が考えられる。しかしここでは、リアルタイム性のないタスクだけのページングを行うだけでも有効性は高いと考えている。

2.3 前提とする環境

前節で述べた提案だけではまだ完全ではなく、2.1.1節に挙げた問題が残る。つまり最悪処理時間が規定可能なページングデバイスを用いるということであるが、本稿では以下の環境を用いてこれに対処する。

- バス結合された分散共有メモリを使用
  我々の研究室で開発した自律移動ロボットアーキテクチャASPIRE[13]は、ロボットの機能毎にプロセッサモジュールを分け、バスで結合した並列計算機アーキテクチャ(図1)である。各モジュールにはメモリが実装されており、相互に参照が可能である。本研究ではこの分散共有メモリをページングデバイスとする。
- ページングを行うモジュールを限定
  ページングを行うモジュールを1つに限定し、その他のモジュール上にあるメモリをページングに用いるようにする。ユーザタスクは

ページングを行うモジュール上で実行する。その他のモジュールはそのモジュールの担当する機能の処理のみ行う。

この環境を用いる利点は、ページングの際にファイルシステムを経由する必要がなく、かつ実行時間を保証するためのディスクスケジューリング等が不要であることである。また不連続領域へのアクセスでも、アクセス時間に大きな変化はないと考えられる。これらの利点から、最悪処理時間の規定が可能であることが期待できる。

2.4 予想される問題点

このページング機構を実装するにあたって予想される問題は、ハードウェア的な制約である。本研究では、バスに関する制約がこれに当たる。2.3節で述べた通り、本研究の実装対象はVMEバスによって各モジュールが結合されたマルチプロセッサである。VMEバスは非同期型の汎用バスであるためにアクセス時間はアクセスの対象となるモジュールに依存するので、リアルタイムシステムには不適当である。またVMEバスはバス獲得とバス割り込みにそれぞれ優先度を持っているが、バスアクセス権の横取りができないために、低い優先度のアクセスが完了するまで高い優先度のアクセスが待たされる状態が存在する。さらにアクセス完了までの時間はアクセス対象のモジュールの仕様に依存するので、アクセス時間の予測が不可能である。

しかしながらASPIREのような、プロセッサだけでなくメモリなども独立したアーキテクチャは、いわば密結合した分散システム[1]と言える。この分散システム上で動作するソフトウェアが、モジュール毎の独立性を高めるソフトウェアアーキテクチャであるならば、バスを通してのアクセス頻度がそれほど高くはならず、最悪アクセス時間を規定することはできると考えられる。そこで、実装対象のVMEバスのアクセス時間を負荷を変えて測定し、それが規定可能性(boundability)を示唆する結果ならば、ページングの処理時間を規定可能にできると考えた。次節ではこれをページング実装のための予備実験として行う。

[1]一般的には分散システムという言葉はネットワークで結合した疎結合モデルを指すが、本稿では密結合モデルを分散システムと呼ぶ。

2.5 予備実験

予備実験の目的は、VMEバスを通したメモリアクセス時間を規定可能とみなせるのか、またバスの負荷によってどの程度変動するのかを調査することである。

2.5.1 実験内容

実験環境および実験内容を説明する。ユーザタスクを実行するメインモジュールはMC68030(25MHz)をプロセッサとするAVME-130-1(AVAL DATA製)であり、RAMを8MB、ROMを512KB実装している。ページングデバイスに用いるモジュールはTMP68301(16MHz)ベースの、我々の研究室で開発したボードである。ROM, 共有RAM, ローカルRAMをそれぞれ1MBずつ実装している。実験にはこのモジュール2枚とメインモジュールの計3枚を用いる。ページング先になるこの2枚のモジュールをページングモジュールと呼ぶことにする。

実験は以下のように行った。ページングモジュールの1つがもう一方のページングモジュールに定常的にデータ転送を行ってバスの負荷を作成し、そこへメインモジュールがページングモジュールに対してデータ転送を実行する。その時の転送にかかった時間をアクセス時間とする。負荷作成ルーチンはアセンブラで作成し、その実行クロックからバスの負荷Lを以下の式により求めた。

$$L = \frac{C_{trans}}{C_{trans} + C_{loop}} \times 100(\%)$$

ここでC_{trans}は実際にバスを通してアクセスしていたクロック数、C_{loop}はループなどの処理のクロック数である。ページングモジュールにはキャッシュが実装されていないので、上式がそのまま適用可能である。負荷の変化はC_{trans}を変えることによって行った。

2.5.2 結果・考察

実験結果を図2に示す。図の各線は、バスの負荷を変えた測定値である。結果には強い線形性が現れた。バスの負荷は0%(無負荷)から65%まで変化させたが、転送にかかった時間の増加は16KB転送時で27%に留まった。量子時間は1msであり、実行時間の規定が可能と言える結果が得られた。負荷の測定は65%までしか行っていないが、先に述べたようにモジュール毎の独立性が高けれ

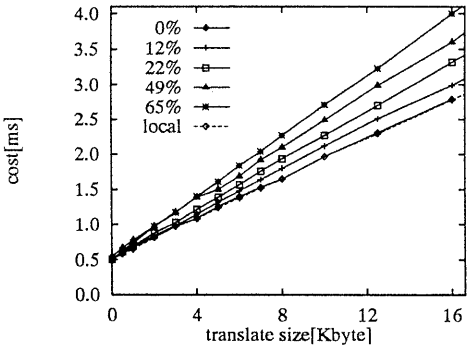

図 2: 転送サイズと転送時間の関係

表 1: ページングに用いるパラメータ

ページサイズ	実行時間
8KB	4ms

ばバスの使用率はここまで高くなることはないと判断し、以上の結果から、ページサイズと量子時間 1ms でのページ転送の最悪実行時間を決定した。

ページサイズには、転送時間が量子時間の 3 倍を越えない最大サイズ 8KB を用いることにした。最悪実行時間は、量子時間の整数倍で転送に要する時間を下回らない値 3ms と、ページのマップその他の処理の時間 1ms の和である 4ms を用いる。表 1にこれらのパラメータをまとめる。以降の実装はこの値を用いて行うことにした。

3 仮想記憶機構の実装

ここでは、最初に実装対象である OS μ-PULSER について説明した後で、実時間ページングの設計と実装について述べる。

3.1 設計

まず実装対象である OS μ-PULSER について説明し、ページング機構の設計方針を述べる。

3.1.1 μ-PULSER について

μ-PULSER は、ASPIRE アーキテクチャ上で動作することを目的とした、ロボット用オペレーティングシステムである [12]。マイクロカーネルアーキテクチャを採用した、マルチタスク・マルチスレッド計算モデルの OS である。カーネル内は必要最小限の機能に抑え、各種サービスをカーネル外の管理スレッドにより提供している。またスレッド間の高速な同期機構が提供されている。

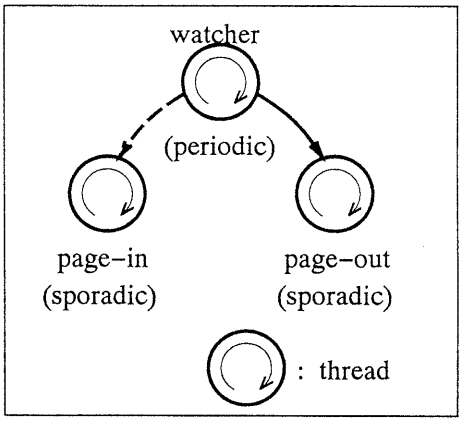

図 3: ページング管理スレッド群

3.1.2 仮想記憶

ページングを実装する前にまず、仮想記憶を実装する必要がある。仮想記憶は 2 つの点に重点を置いて設計した。1 つはタスク毎のメモリ空間の分離である。仮想記憶を実装する前はアドレス空間は単一で、すべてのメモリは共有されていた。これをタスク毎に論理的に独立させ、カーネル空間、I/O 空間、そして通信用バッファのみを共有するようにした。これによってアドレス空間の保護を実現し、かつ同期機構によるタスク間にまたがるスレッド間通信を可能とした。もう 1 つはページ管理についてである。2.2.3節で述べたように、ページアウトはリアルタイムでないタスクのページのみに限定する。そのためページ管理構造体に**ページアウト禁止フラグを設ける**ことにした。

仮想記憶の管理はすべて、管理スレッドの 1 つであるメモリマネージャが行う。μ-PULSER ではすべての管理スレッドはユーザ状態で動作するが、仮想記憶の操作には特権状態で行わなければならない処理 (ATC エントリのフラッシュなど) があるので、そのためのシステムコールを追加した。

3.1.3 ページング管理スレッドの設計

ページングはメモリ管理と同様に、管理スレッドとして実装した。ただし 1 つのスレッドではなく、各処理毎にスレッドを分け、複数のスレッド全体でページングを実現することにした。すなわち、(1) 空きメモリ監視スレッド、(2) ページアウ

ト実行スレッド、(3) ページイン実行スレッドの3つである。

- **空きメモリ監視スレッド: watcher**
  リアルタイムスケジューラによって周期的に起動され、空きメモリ量を調べる。その量に応じてページアウトスレッドまたはページインスレッドに同期を取り、ページングを行う。また起動されるたびに一定数のページの使用ビットをクリアする。これはLRUアルゴリズムのためで、UNIXの「2本針アルゴリズム」とほぼ同様である。ページアウトの際には使用ビットのついていないページがページアウトの候補となる。
- **ページアウト実行スレッド: pageout**
  watcherによって起動される非周期的スレッドであり、普段は眠っている。LRUアルゴリズムによるページアウト候補からページを選択し、追い出し先の共有メモリのアドレスを選択して追い出す。実行時間には2.5.2節で述べた4msを設定する。
- **ページイン実行スレッド: pagein**
  pageoutと同じくwatcherに起動される非周期的スレッドである。FIFOに従ってページアウトされているページを選択し、再びメモリにマップする。またページフォルト発生時には、フォルトを起こしたページをページインし、もとのコンテキストに戻る。

これらのスレッドの関係図を図3に示す。一番上に書いてあるスレッドがwatcherである。空きメモリ S_{free} が S_{page_out} より少なくなると、実線の矢印のようにwatcherがpageoutと同期を取り、ページアウトを行う。反対に空きメモリ S_{free} が S_{page_in} より大きくなると、破線の矢印が示すようにwatcherがpageinに対して通信し、ページインを実行する。これらのスレッドはユーザタスクと直接通信することはなく、ユーザタスクとのメモリのやりとりはみな先に述べたメモリマネージャが行う。メモリ確保の際に深刻なメモリ不足が生じた時は、メモリマネージャがpageoutにメッセージを送って緊急にページアウトを実行する。唯一の例外はページフォルトが発生した時で、この時はメモリマネージャを通さずに直接pageinコンテキストに移ってページイン処理を行う。

表 2: 実装に用いるパラメータ

ページサイズ	8KB
ページング実行時間 C	4ms
watcher 周期	250ms
S_{page_in}	256KB
S_{page_out}	128KB

3.2 実装

以上に述べた設計方針に従って、実装を行った。実装するモジュールは予備実験と同じMC68030(25MHz) をプロセッサとしたAVME-130-1である。RAMは8MB, ROMは512KBである。実装にあたって決めるべきパラメータは、ページサイズ、watcherの周期、ページングスレッドpageinとpageoutの実行時間 C、およびページングを起動する閾値 $S_{page\_in}, S_{page\_out}$ であった。表2に決定したパラメータの一覧を示す。ページサイズとページングスレッドの実行時間 C は予備実験から得た値を用いた。watcher 周期は4.3BSDのページアウトと同じに設定した。閾値に関しては強い根拠はないが、ページアウトの閾値を実メモリの2%程度、ページインを実メモリの5%程度と設定して決めた。

4 評価・検討

ここでは、今まで述べたページング手法がリアルタイムシステムで動作可能なのかどうかを調べるために、ページアウトにかかった時間、およびページフォルト時のハンドリングに要した時間を測定した。

4.1 ページアウトの処理時間

リアルタイムスケジューラの下では、ページアウトの処理が確実にデッドラインまでに終了しなければならない。しかも周期的負荷の変動やバスの負荷の変化にも耐えられなければならない。これを確認するためにページアウトの処理時間を、バス負荷、周期的負荷を変えて測定した。バスの負荷は予備実験と同じ方法で作り、周期的負荷は周期50ms、実行時間1msの周期的スレッドの数を変化させて作成した。飯田らによって、sporadic serverアルゴリズムが非周期的タスクのスケジューリングに適していることが示されている [3] ので、本研究ではスケジューリングアルゴリズムにsporadic

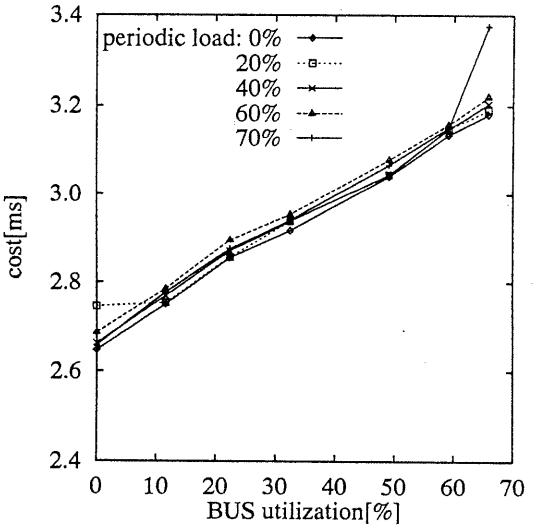

図 4: バスの負荷に対するページアウトの処理時間

server アルゴリズムを選択した。図 4にその結果を示す。バス負荷の増加に伴って処理時間が増加しているがその増加分はわずかであり、規定可能性を示唆している。設定した最悪実行時間は 4ms なので、測定した範囲内では全てスケジューラの行うスケジューリングより早く処理が終了している。バスの負荷 65%、周期的負荷 70%の時に処理時間が急激に増加しているが、それでも十分早く終了しているので、スケジューリングに与える影響はない。

興味深いのは、周期的負荷の変動に非常に強い点である。0%から 70%まで変化させたが、処理時間にほとんど違いは見られなかった。これはページング処理時間である 4ms が周期的タスクの周期 50ms のわずか 8%であり、しかも最悪実行時間より早く終了しているためであると考えられる。しかし、周期的負荷を 75%にした場合、デッドラインミスが発生してスケジューリング不可能になった。この結果から、評価に用いたスケジューラのブレイクダウンユーティライゼーションは 75%+8% = 83%であると言える。一般に、ページングが実行可能な最大プロセッサ利用率はスケジューラのスケジューリングアルゴリズムに依存する。従ってページング機構だけでなく、スケジューラも含めた設計が重要であろう。

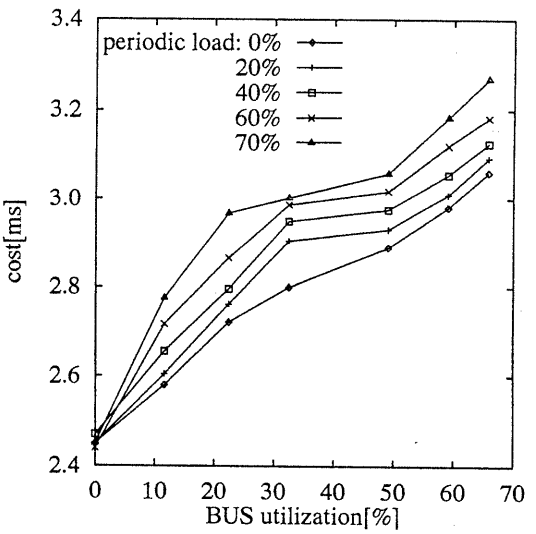

図 5: バスの負荷に対するページフォルトの処理時間

4.2 ページフォルトの処理時間

ページフォルトを起こさないように watcher を設計するのが基本方針ではあるが、実際にはページフォルトが発生する状態が十分考えられ、しかもその際には規定可能性が非常に重要となる。そのためページフォルトのハンドリング時間を測定した。測定は、あらかじめ強制的にページアウトさせておいたページに対しての書き込みをすることによって行った。データはページアウトの測定と同じく、バスの負荷および周期的負荷を変化させて測定した。この結果を図 5に示す。

ページフォルトでは、ページアウトの場合より周期的負荷によってばらつきが出た。しかし、その差は最大でも 240μs であり、設定した実行時間中に十分処理が可能であった。ただページアウトで強く現れていた線形性が薄れており、これは今後の検討課題である。

ページフォルトが発生した場合、そのタスクの実行時間は図 5に示した処理時間だけ長くなる。しかし、積極的なページングによってページフォルトが起きる前にページがマップされていれば、処理時間の悪化を避けられる。この積極的なページングの性能は、ページングを起動する閾値 $S_{page\_in}, S_{page\_out}$ に大きく関係していると考えられる。この閾値は、今回は指標にすべきデータが

なかったので詳しく検討せずに設定したが、その最適値はおそらく、実行するタスクセットや動作環境に大きく依存することが予想される。閾値の最適な設定は、積極的なページングの評価と合わせて今後の課題である。

5 まとめ

リアルタイムOSにページングを実装するための手法を提案し、分散共有メモリをページングデバイスとして用いて実装した。ページング機構は複数のスレッドに分け、より積極的なページングを行うように実装した。評価の結果、バスの負荷や周期的負荷の変動にさほど依存しないページング処理時間が得られた。

一方で、ページングが実行可能な最大プロセッサ利用率はスケジューリングアルゴリズムに依存するため、ページングアルゴリズムとスケジューラ双方からの検討が必要であることが分かった。今後はこれらの検討を進めていく予定である。

参考文献

[1] Anderson D. P., Ozawa Y., and Govindan R. : A file system for continuous media, *ACM Transactions on Computer Systems*, Vol. 10, No. 4, pp. 311–337, November 1992.

[2] 今村博宣：VMEバスの規格と設計のポイント，インターフェース，pp. 202–221. CQ出版社，August 1985.

[3] 飯田浩二，矢向高弘，菅原智義，安西祐一郎：スケジューリングポリシの動的変更に関する研究，実時間処理に関するワークショップ(RTP'93) 情報処理学会研究報告 93-ARC-99, pp. 70–76, March 1993.

[4] Leffler S., McKusick M., Karels M., and Quaterman J. : *The Design and Implementation of the 4.3BSD UNIX Operating Systems*, Addison-Wesley Publishing, 1989.

[5] Li K. and Hudak P. : Memory Coherence in Shared Virtual Memory Systems, *IEEE Transactions on Computer Systems*, Vol. 7, No. 4, November 1989.

[6] 西田健次，戸川賢二：実時間処理のためのハードウェアアーキテクチャ，情報処理，Vol. 35, No. 1, pp. 26–33, January 1994.

[7] Park C. Y. : Predicting program execution times by analyzing static and dynamic program paths, *Real-Time Systems*, Vol. 5, No. 1, pp. 31–62, March 1993.

[8] Rangan P. V. and Vin H. M. : Designing file systems for digital video and audio, In *Proceedings of the Thirteenth ACM symposium on operating systems principles*, pp. 81–94. Asilomar Conference Center, October 1991.

[9] Sprunt B., Sha L., and Lehoczky J. : Aperiodic task scheduling for hard-real-time systems, *The Journal of Real-Time Systems*, Vol. 1, pp. 27–60, 1989.

[10] Stankovic J. A. : Distributed real-time computing: The next generation, 計測と制御, Vol. 31, No. 7, pp. 726–736, July 1992.

[11] Tokuda H., Nakajima T., and Rao P. : Real-Time Mach: Towards a Predictable Real-Time System, In *Proceedings of USENIX Mach Workshop*, pp. 1–10, 1990.

[12] 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，February 1994.

[13] 山崎信行：パーソナルロボット用アーキテクチャASPIREの設計と実装，計算機科学専攻修士論文，慶應義塾大学，March 1993.