オペレーティング・システム 51－8
（1991. 7. 18）

分散オペレーティングシステム $R^2/V3$ における分散共有メモリの実現

國枝 和雄[†] 長野 晋[††] 大久保 英嗣[†††] 津田 孝夫[†]

[†] 京都大学工学部情報工学科
[††] ソニー(株)
[†††] 立命館大学理工学部情報工学科

分散OS $R^2/V3$ における分散共有メモリの実現手法について述べる。$R^2/V3$ では、タスクグループを構成することによって、タスクはグループ内の他のタスクとメモリ空間を共有することができる。各共有メモリは一つのセグメントとして実現されており、セグメント単位でのハードウェアによるメモリ保護を行なうことができる。また、これに加えてタスクグループに基づく論理的な保護も行なわれる。我々は、このような特徴を持つ分散共有メモリを $R^2/V3$ の基本機能として実現した。さらに、性能測定を行ないその有効性を検証した。

An Implementation of Distributed Shared Memory in Distributed Operating System $R^2/V3$

Kazuo Kunieda, Susumu Nagano, Eiji Okubo, Takao Tsuda

Department of Information Science, Faculty of Engineering, Kyoto University
Yoshida hon-machi, Sakyo-ku, Kyoto 606, Japan

In this paper, an implementation of distributed shared memory in the distributed operating system R2/V3 is described. In R2/V3, by grouping tasks, a task can share the memory space with other tasks within the group. Each shared memory is formed into a segment, so hardware memory protection for each segment is possible. Furthermore *task group* based logical protection is also possible. The shared memory system with above features is developed as a basic function of R2/V3. In addition, we evaluate the performance and inspect the effectiveness of it.

1 はじめに

ネットワークに分散した計算機資源を有効に利用するために、様々な分散OSが開発されている。分散OSでは、ユーザからネットワークやハードウェアの構成を隠蔽する機能、すなわち透過性が提供される。この中で特に重要なのがメモリ資源の透過性である。メモリ資源の透過性が実現されると、サイトの内外を問わず複数のタスク間で直接的なデータの共有が可能となり、タスク間通信やデータ処理を効率的に行うことができる。また、ネットワーク内でのアドレス空間の不透明さが除去され、タスクのネットワーク透過な実行が可能となるなど、他の資源の透過性の実現が容易になる。

このメモリ資源の透過性を実現する手法の一つとして、分散共有メモリがある [1]。現在実現されている分散共有メモリの多くは、各サイトで全く同じアドレス空間を定義することによってメモリの透過性を実現している場合が多い [2, 3, 4]。このようなアプローチでは、全サイトの全タスクがアドレス空間を共有することしかできない。本来、タスクのアドレス空間はそれ自身で閉じており、他のタスクの空間とは独立している。他のタスクとアドレス空間の全体あるいは一部を共有する場合には、あらかじめ許可されたタスクのみ共有領域にアクセス可能とするような制限を設けるべきである。

したがって、タスクのアドレス空間の安全性が図られたメモリの透過性の実現のためには、ネットワーク上の任意のタスクが、そのアドレス空間の一部として、他のタスクとの共有領域を持つことが可能であるような分散共有メモリ機能が必要である。そこで、我々はこのような分散共有メモリを分散OS $R^2/V3$ の機能として実現した。$R^2/V3$ では、ハードウェアの提供するメモリ保護機能を最大限に利用するとともに、タスクグループの概念を導入し、タスクグループ間での論理的な保護機能を実現している。また、共有メモリ操作間の同期をとるための分散セマフォも実現した。このように、$R^2/V3$ の分散共有メモリはユーザタスクに対して柔軟性の高い機能を提供している。

本稿では、$R^2/V3$ における分散共有メモリ機能の概要と実現手法について述べる。さらに、並列分散処理における操作性や処理速度などの観点から、その性能について検討する。

2 分散共有メモリ

2.1 管理方式

分散共有メモリは、ネットワーク上の異なるサイトのローカルメモリを一つの論理的な実体に統合し抽象化したものである。各サイトのローカルメモリはキャッシュとして扱われ、タスクのアドレス空間はサイトの境界を越えている。各サイトでローカルなメモリに存在しないデータにアクセスしようとするとネットワークページフォルトが発生し、必要なメモリイメージがリモートサイトから転送される。ここで、共有メモリの管理を行なうサイト (*manager* と呼ぶ) の配置について、次の二通りの方法が考えられる。

(1) 静的管理方式

共有メモリ空間の管理を、ある特定のサイトで一括して管理する方式である。あるいは、共有メモリ空間をいくつかに分割して、それぞれの分割単位を異なるサイトで管理する方式も考えられる。*manager* の配置が固定されているので、*manager* に対して直ちにメモリの転送要求を出すことができる。しかし、共有メモリのイメージ自体は、最新のメモリイメージを確保しているサイトが要求元サイトへ転送しなければならない。したがって、この方式ではネットワークへのアクセス数 (メッセージ数) が多くなる。

(2) 動的管理方式

共有メモリへの書き込み操作に伴って *manager* が移動する方式である。その後のメモリ要求には、最後に書き込みを行ったサイトが *manager* として応えることになる。この方式では *manager* の配置が固定されないので、常にその配置に関する情報

の更新を行わなければならない。しかし、共有メモリイメージの転送は、*manager*から直接要求元に向けて行われるので、ネットワークへのアクセス数は少ない。

一般に、ネットワークを介したメッセージ通信は大きな遅延を伴うことから、$R^2/V3$では共有メモリの管理を動的管理方式で行っている。

2.2 一貫性制御方式

他のサイトに存在する共有メモリに読み出し要求を行なった場合、その内容は転送され、ローカルなメモリにコピーされる。このとき共有メモリのコピーが複数サイトに存在することになる。したがって、分散共有メモリにおいても、密結合マルチプロセッサシステムにおけるキャッシュ一貫性(coherence)の制御と同様に、各サイトのメモリの一貫性を保証しなければならない。すなわち、任意のタスクが共有メモリから読み出す値は、そのアドレスに対して行われた最後の書き込み操作の値でなければならない。密結合システムでは、ローカルキャッシュへの書き込み操作に対して、主記憶への値の反映と他のキャッシュの無効化という操作で一貫性を保証するのが一般的である。一方、分散共有メモリの場合には、バックボーンとなる主記憶は存在しないので、書き込み操作時には、他サイトのメモリに対する値の反映と無効化が必要となる。これについて以下の二つの制御方式を考えることができる。

(1) ライトスルー (write-through) 方式

共有メモリへの書き込みが起こった場合、同時に他サイト上の共有メモリのコピーに対しても書き込みを行う。この方式では無効化は必要ない。

(2) 無効化 (invalidation) 方式

共有メモリへの書き込みが起こった場合、まず、他サイトの共有メモリのコピーを無効化するように各サイトにメッセージを送る。そして、他サイトで読み出しが行なわれた時点で、改めてメモリの内容を転送する。

ライトスルー方式は、各サイトの共有メモリを常に最新の状態に保つことができるが、1バイトあるいは1ワードといった小さな単位での書き込みが発生する度に各サイトにメッセージを送らなければならない。したがって、メッセージ通信によるオーバヘッドを無視することのできない分散システムには不適当である。しかも、密結合システムのように、特別なハードウェアによる支援がなければ分散システム上では実現が困難である。したがって、$R^2/V3$では、無効化方式を採用している。

2.3 メモリ属性

共有メモリに対する書き込み・読み出しを行なうためには、プロセッサは書き込み権・読み出し権を獲得しなければならない。ある時点において、書き込み権を持つサイトはただ一つ存在し、読み出し権を持つサイトは複数存在し得る。ここで、書き込み権を持つサイトを*owner*と呼ぶ。書き込み件は、メモリへの書き込み要求によってサイトを移動する。読み出し権は、メモリの無効化によって剥奪される。

$R^2/V3$では、各サイトの共有メモリに、read-write、read-only、invalidの三つの属性のうちどれか一つを指定し、一貫性制御を行っている。各属性の意味は以下の通りである。

(1) read-write

共有メモリの内容は最新のものである。この属性のコピーはシステム内に高々一つ存在し、これを持つサイトが*owner*となる。また、$R^2/V3$が採用している動的管理方式では、この属性のコピーが存在するサイトが*manager*サイトにもなる。

(2) read-only

共有メモリの内容は最新のものである。読み出し権を持つタスクは、この共有メモリ内のデータを読み出すことができる。しかし、書き込みが行われると書き込み違反となる。

(3) invalid

共有メモリのコピーは無効である。タスクがこのメモリにアクセスすると、その種類に応じて書

き込みあるいは読み出し違反が発生する。

書き込みおよび読み出し違反は、共有メモリの一貫性を保証するために引き起こされるものでありメモリ保護のためのものではない。したがって、書き込み違反を起こしたサイトは書き込み権を、読み出し違反を起こしたサイトは読み出し権を与えられなければならない。各場合の処理を以下に示す。

(1) 書き込み違反の場合

(a) *owner* に共有メモリへの書き込み許可を依頼する。

(b) *owner* は、最新のメモリイメージとコピーの存在するサイトの情報を返信する。

(c) 書き込み違反を起こしたサイトは、返信メッセージにしたがって無効信号を送信し、*owner* 交替の通知とコピーの無効化の要求を行う。

(d) 送信した全てのサイトから無効化の返信を受け取った後、共有メモリの属性を read-write に設定してユーザタスクに制御を戻す。

(2) 読み出し違反の場合

(a) *owner* に共有メモリへの読み出し許可を依頼する。

(b) *owner* は最新のメモリイメージを返信する。

(c) 読み出し違反を起こしたサイトは、返信されたコピーの属性を read-only に設定し、ユーザタスクに制御を戻す。

2.4 アクセス制御

共有メモリは複数のタスクが直接参照することができるという点が大きな特徴であが、無制限にアクセスを許すことは適当でない。マルチタスクシステムでは、複数のタスクが、協調し合いながら、あるいは全く独立に処理を行っている。共有メモリを他のタスクとの交渉手段として用いる場合、それは限られた相手にのみ提供されるものでなければならない。また、故意によるものでなくても、例えばプログラムの誤りから、他のタスクが使用している領域を破壊してしまうようなことがないようにしなければならない。

$R^2/V3$ は、以下に挙げるような手段を講じることによって、共有メモリへのアクセス制御を行なっている。

(1) 名前による制御

$R^2/V3$ では、共有メモリへのポインタを得るためには、共有メモリ作成時に登録された名前を指定しなければならない。これにより、共有メモリの存在を、あらかじめ定められたタスク以外のタスクに知られることを防いでいる。また逆に、他サイトで生成されたタスクであっても、名前を指定することにより、共有メモリへのアクセスが可能になっている。

(2) グループ識別子による制御

$R^2/V3$ では、システム内の任意のタスクの間でタスクグループを形成することができる。共有メモリ作成時にそれを利用するタスクの集合を表すグループ識別子を指定することによって、グループ以外のタスクからの共有メモリの利用を制限することができる。したがって、ユーザが能動的に共有メモリのアクセス制御を行うことができる。またこの制御によって、故意、あるいは偶然によって名前が同一の共有メモリが作成された場合にも正確なアクセス制御を行うことができる。

(3) セグメントによる保護

共有メモリは、ユーザによって指定された大きさのセグメントとして定義される。したがって、ポインタ操作などによって、セグメント外の領域をアクセスするようなことが起こっても、ハードウェアによってセグメント保護機能が働き、他のタスクに影響が及ぶことはない。

2.5 同期制御

分散共有メモリは、複数のサイトに分散した共有メモリのコピーの一貫性を保証している。しかしこれは、任意の時点で各サイトの共有メモリの内容が完全に一致していることを保証したものであり、複数のタスクによる同一場所への値の代入といった共有メモリへの並行アクセスに関して、その逐次実行性 (serializability) を保証するものではな

い。したがって、並列プログラミングを支援するために、同期制御のための基本命令が必要となる。

$R^2/V3$では、タスク間の同期制御のために分散セマフォを提供している。

3 ユーザインタフェース

$R^2/V3$において、共有メモリを使用するためのユーザインタフェースについて述べる。ユーザインタフェースは以下の三つのSVCとして提供される。

(1) 共有メモリの作成

$R^2/V3$において、ユーザは次のSVCを用いて共有メモリを作成する。

```
void *
createSharedMemory(gID, size, name)
    int gID;
    int size;
    char *name;
```

これにより、nameという名前のsizeバイトの大きさの共有メモリが仮想アドレス空間上に確保され、それへのポインタが、戻り値としてユーザに返される。

gIDは作成するメモリを利用するタスクの集合を表すグループ識別子である。作成された共有メモリは、gIDで特定されるグループに属しているタスクだけが参照することができる。gIDとしてALL_TASKを指定した場合、生成された共有メモリは、全てのタスクから参照可能となる。

(2) 共有メモリのアドレスの取得

共有メモリを作成したタスク以外のタスクは次のSVCを用いて共有メモリへのポインタ(アドレス)を獲得する。

```
void *
getPtr2SharedMemory(name)
    char *name;
```

ただし要求元タスクが、名前nameで示される共有メモリへのアクセスを許されているグループに属していないならばゼロが返される。$R^2/V3$では、値がゼロであるポインタを用いたメモリアクセスは例外を引き起こし、実行中のタスクは強制終了させられる。

一旦ポインタを獲得すれば、ユーザはそれを用いて自由に、そして他のメモリと何ら区別することなく共有メモリにアクセスすることができる。

(3) 共有メモリの解放

共有メモリが不要となった場合は、次のSVCを用いて共有メモリを解放することができる。

```
void
freeSharedMemory(name)
    char *name;
```

実際には、getPtr2SharedMemory()を発行し、共有メモリへのアクセスが許されているタスクのすべてがこのSVCを発行するか実行を終了するまで、その共有メモリはメモリ上に残される。また、共有メモリを使用するタスクが全て終了すれば、自動的に共有メモリの解放を行う。

4 ハードウェアインタフェース

現在、$R^2/V3$はCPUとしてIntel80386を持つマシン用に開発されている。ここでは、特にネットワークページフォルト処理を80386上で実現する手法について具体的に説明する[5]。

4.1 メモリ空間の管理

$R^2/V3$の各サイトにおけるローカルメモリ空間は、ページングによって実現されるリニアアドレス空間(4GB)である。その上に各セグメントを配置することによってメモリを管理している。一方、タスクにおけるメモリの基本構成は、テキスト、データ、スタックの3セグメントである。タスクが共有メモリを利用する場合、共有メモリセグメントがこれに加えられる。各共有メモリはそれぞれ独立のセグメントとして実現されており、複数の共有メモリを利用する時には、その数だけのセグメントが加えられることになる。

80386においてタスクからのセグメントアクセスは、セグメントディスクリプタをタスク

のローカルディスクリプタテーブル (LDT) に登録することによって可能となる。したがって、createSharedMemory() によって共有メモリの作成要求があると、共有メモリ管理部は、ユーザが指定したサイズのセグメントをリニアアドレス空間上に定義し、そのディスクリプタを LDT に登録する。共有メモリのディスクリプタを作成する場合に必要となる情報は、共有メモリのベースアドレスとサイズである。$R^2/V3$ では、それらの情報をテーブル SHM_tbl に格納している。SHM_tbl にはこの他、共有メモリの名前、*owner* サイト、コピーの存在するサイト、アクセス可能なグループのグループ識別子などの情報が格納されている。

createSharedMemory() で生成された共有メモリは、その SVC を発行したタスクのローカルなアドレス空間内にあるため、他のタスクからは "見えない"。これを "見える" ようにするのが getPtr2SharedMemory() である。getPtr2SharedMemory() は、共有メモリのアドレス空間を、自分のアドレス空間の一部に取り込むための SVC である。$R^2/V3$ では、アドレス空間の共有は、共有したい仮想アドレス空間上の領域を指し示すディスクリプタを作成するだけで容易に実現できる。getPtr2SharedMemory() が発行されると、OS の共有メモリ管理部は SHM_tbl から、SVC の引数で指定された名前を持つ共有メモリを探し出し、その項目に格納されているベースアドレスとサイズを使用して、SVC を発行したタスクの LDT に共有メモリのディスクリプタを作成する。該当する共有メモリが SHM_tbl に登録されていない場合は、createSharedMemory() の処理と同様に、共有メモリのために新たに仮想アドレス空間を確保しなければならない。ただしこの場合、サイズをはじめ共有メモリに関する情報が存在しない。従って、共有メモリ管理部はブロードキャストによって他のサイトに問い合わせを行い、共有メモリの管理サイトからの返信を基に、ディスクリプタの作成及び自サイトの SHM_tbl への登録を行う。一旦共有メモリが自サイトの SHM_tbl に登録されれば、以降は、共有メモリが生成されたサイトと通信することなく getPtr2SharedMemory() を処理することができる。createSharedMemory() によって生成される共有メモリの初期属性は invalid である。また、他サイトで生成された共有メモリに対する getPtr2SharedMemory() の結果生成される共有メモリの初期属性も同様に invalid である。即ち、確保された共有メモリに対して、実際には物理メモリは割り当てられず、最初の書き込み、あるいは読み出しが行われた時に初めて物理メモリが割り当てられる。これは、SVC の処理に要する時間を短縮するためと、メモリ資源を有効に利用するためである。

4.2 ページフォルトハンドリング

一貫性制御に伴うネットワークページフォルトに対する処理は、CPU のメモリ保護機能を利用して行なう。すなわち、共有メモリへのアクセスを CPU におけるメモリ保護例外として検知する。検知しなければならないアクセスは、read-only 属性の共有メモリへの書き込みと、invalid 属性の共有メモリに対する書き込み及び読み出しである。80386 にはセグメント単位の保護機構とページ単位での保護機構があり、それらを利用することが考えられる。$R^2/V3$ では、共有メモリをセグメントとして実現しているので、セグメント単位のメモリ保護機能を利用するのが最適であると考えられる。しかし 80386 のセグメント単位の保護機能では、保護例外が発生した時にその要因が書き込みと読み出しのいずれであるか判定できない。そこで、$R^2/V3$ では 80386 のページ単位の保護機能を利用している。

この制御のために、ページテーブルエントリを利用して 2.3節で示した共有メモリの属性を表現する (表 1を参照)。

ページの属性は、ページエントリ内の P ビット (Present bit)、R/W ビット (Read/Write bit) および swap_page_no フィールドで表される。P ビットによって共有メモリの主記憶上での存在がわかる。しかし、P ビットはページングにも使われているため、共有メモリがページアウトされている場合にも P=0(ページ不在) になる。そこで、P=0

表 1: 共有メモリとページの属性及びアクセス違反

共有メモリの属性	ページの属性			アクセス違反	
	P	R/W	swap_page_no	種類	エラーコード
read-write	1	1	SOMEWHERE	—	—
read-only	1	0	SOMEWHERE	書き込み	111
invalid	0	×	NOT_ALLOCATED	書き込み 読み出し	110 100

(×は不定を表す)
(SOMEWHERE は P=0 の時 NOT_ALLOCATED でないことを表す)
(エラーコードは下位 3 ビットを示す)

の場合に invalid なのかページアウトされているのかを区別するために swap_page_no を用いる。このフィールドは、本来、ページの二次記憶上の退避ページ番号を格納するためのものである。invalid 属性は、ここに特殊な値をセットすることで区別される。また、共有メモリが複数のページにまたがる時は、それら全てのページに同じ属性を設定する。その後ページフォルトが発生すると、その原因を調べる必要がある。$R^2/V3$ はそのために、80386 によってユーザタスクのスタックにプッシュされるエラーコードを用いる。

ここで、ページの転送を依頼された *owner* サイトの属性を read-only に設定する処理は、書き込み要求だけでなく読み出しの要求を受け取った時にも行わなければならない。なぜならば、*owner* サイトの属性を read-write のままにしておくと、*owner* での書き込み対して例外が発生せず、その内容を他のコピーに反映させることができないからである。

4.3 メッセージ交換手順

分散システムではメッセージの到着順序に関する仮定を立てることはできない。したがって、メッセージの到着に関して、予め考えられ得る全ての組合せについて検討し、処理に論理的な誤りがないことを保証しなければならない。以下に、$R^2/V3$ におけるサイト間通信を実現するにあたって考察した点をいくつか述べる。ここで、書き込み権を持つサイトを *owner*、そのコピーを持つサイトの集合を *copy_sites* とする。

(1) 共有メモリの *owner* の正確性

SHM_tbl 内の latest_site フィールドは、共有メモリの現在の *owner* 番号を格納している。*owner* の交替は、現在の *owner* が他サイトからの書き込み要求 (WRITE_REQ) を受けとった時に起こる。したがって、ある時点では、各サイトの latest_site の値は間違った値となる。しかし、新しい *owner* が *copy_sites* の全サイトに無効化メッセージ (INVALID_REQ) を送信し、メモリ内容とともに latest_site を無効化することによって各サイトは latest_site の値を正しく変更することができる。

(2) 共有メモリのネットワーク内探索時の通信

ある共有メモリの *owner* サイトを特定するためのメッセージ WHERE_REQ に対する返信は *owner* 自身が行う。ここで、動的管理方式によって *owner* は他サイトで書き込みが行われる度に変化するため、WHERE_REQ の返信が要求元に到着した段階ですでに *owner* が移動している可能性がある。$R^2/V3$ ではこれを解決するために、WHERE_REQ を受けとった *owner* は、要求元サイトから WHERE_REQ_END メッセージが届くまでその後の WRITE_REQ の受付を保留する。WHERE_REQ_END は、WHERE_REQ の要求元サイトが返信を受け取り、その結果必要となる処理を行なった後で、*owner* の移動を許可するため

のメッセージである。

(3) 読み出し要求中の無効化メッセージの受信

読み出しのために共有メモリのコピーを *owner* に要求している時、その共有メモリの無効化の依頼 (INVALID_REQ) が他のサイトから来ることが考えられる。この場合には再度 READ_REQ メッセージを発行する。処理量、通信路のトラフィックなどを考慮して、以前に READ_REQ メッセージを送信したサイトに取消を指示するようなメッセージは送信しない。

5 評価

$R^2/V3$ 上に実現された分散共有メモリの性能測定の結果を示し、その評価を行う。

実験は、サイトとして PC-9801RA2、LAN として Ethernet を使用して行った。サイト数は 4 である。測定は、共有メモリに対する最初のアクセスの結果起こるページフォールトに対し、それが発生してから制御がユーザに戻るまでの時間について行なった。また、共有メモリに対する読み出しと書き込みのそれぞれについて、共有メモリのサイズを 256 バイトから 4096 バイトの 6 通りに変化させて測定を行った。表 2に実験結果を示す。

表 2: メモリアクセスに対する処理時間

SIZE	256	512	1024	2048	3072	4096
READ	165.6	173.9	205.9	236.3	281.3	363.6
WRITE	396.6	379.8	383.3	417.9	485.3	536.4

(単位: msec)

表から分かるように、共有メモリのサイズとともに通信時間も増加する。書き込みに対する処理に必要な時間は読み出しに対する処理に必要な時間より多いのは、無効化の確認をするためのメッセージ通信を必要とするためである。$R^2/V3$ が使用しているサイト間通信に要する時間は約 15msec/ 回である。したがって、実験結果については、一貫性制御のために行なわれる通信のオーバヘッドがかなりの部分を占めているものと思われる。現在、性能を改善するために、より高速な通信方式の採用について検討中である。

6 おわりに

本稿では、分散型実時間 OS $R^2/V3$ における分散共有メモリの実現技法について述べた。

分散共有メモリによって、ネットワーク上の複数のタスク間での直接的なデータの共有を可能にした。これによって透過性を持つメモリ資源が実現されたといえる。タスクのアドレス空間全体の透過性を実現し、ネットワーク中のすべてのメモリに透過的にアクセスできるようにすることが今後の課題である。

参考文献

[1] Ming-Chit Tam, Jonathan M. Smith, and David J. Farber: A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems, Operating Systems Review, Vol. 24, No. 3, pp. 40–67(1990).

[2] Kai Li and Richard Schaefer: A Hypercube Shared Virtual Memory System, Proceedings, International Conference on Parallel Processing, Vol. I, pp. 125–132(1989).

[3] Kai Li: IVY:A Shared Virtual Memory System for Parallel Computing, Proceedings, International Conference on Parallel Processing, Vol. II, pp. 94–101(1988).

[4] Ronald G. Minnich: The Mether:Distributed Shared Memory for SunOS 4.0, Proceedings, USENIX–Summer Conference, pp. 51–60(1989).

[5] intel Corporation: 80386 System Software Writer"s Guide(1987).