2003－OS－94　(2)
2003／8／4

複数実計算機における非共有リソース利用方式の実装

桝本　圭†　　田渕　正樹†　　伊藤　健一†
乃村　能成††　　谷口　秀夫‡

一つのプロセッサ上に複数の OS を同時に走行させる技術が普及している．仮想計算機方式は，この技術を実現する既存方式であり，一つの主たる OS 上で他の OS がアプリケーションとして動作する形式である．このため，主たる OS の停止が全体の停止を招く問題がある．そこで我々は，複数の OS が独立に走行するための複数 OS 構成法を提案した．本構成法では，各 OS の独立性を高めるために，ハードウェアは各 OS に占有させる．しかし，ハードウェアが複数用意できない場合など，状況に応じて特定のハードウェア OS 間で共有することにより，利便性を高めることも必要である．そこで，本稿では，ハードウェア資源を個別の OS に占有させつつ，特定のハードウェア資源だけを安全に，かつ選択的に共有させる方式を提案する．また，本方式を PCI ハードウェアに適用する方法についても述べる．

Implementation of a Hardware Sharing Method for Multiple Real Machine on a Single System

Kei MASUMOTO†, Masaki TABUCHI†, Ken-ichi ITOH†,
Yoshinari NOMURA†† and Hideo TANIGUCHI‡

Multiple operating system has become popular. Major existing method is that secondary operating systems run on a primary operating system. This relation between primary operating system and others causes some problem such as an accidental stop of primary operating system leads to the others. We proposed a new framework that operating systems can run independently and simultaneously on a single processor. In basic concept of this framework, coexistent operating systems don't share hardwares for their independency. But for better availability, it is desirable that they can share hardwares. This paper describes a method for sharing hardware resource among the coexistent operating system and its implementation focused on PCI hardware. In this method, each operating system occupies some hardwares but can provide arbitrary hardware resource to the others safely.

１．はじめに

パーソナルコンピュータ（以降 PC と呼ぶ）の普及により，様々な特徴をもつ OS が現れた．そして，これらの OS を同時に稼動させることにより，複数の OS の長所を合わせ持つことを実現する技術への要求が高まった．

複数 OS を実現するために，従来は一つの OS 上で他の OS をアプリケーションのように稼動させる実現方式（仮想計算機）を採っていた．

仮想計算機の問題点として，一つの OS が自身のメモリ空間やハードウェア上で，他方の OS を稼動させるため，互いに悪影響を及ぼすことがある．

この問題を解決するために，我々は，各 OS が使用するハードウェアを分割し，互いの OS の独立性を高める複数 OS の構成法である複数実計算機（Multiple Real Machines：MRM）を提案した[1]．MRM の特徴は，各 OS が独立に稼動するため，互いに悪影響を及ぼす問題を軽減することである．

MRM の基本的な考え方では，各 OS は自分が占有していないハードウェアを利用できない．しかし，計算機に同種のハードウェアを複数持つことは避けたい場合や，ハードウェアを二つの OS で使用すると利点が生じる場合には，二つの OS にハードウェアリソースを共有させたい．

そこで，ここでは一つの OS が占有するハードウ

†株式会社 NTT データ技術開発本部
Research and Development Headquarters, NTT DATA Co.
††九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University
‡岡山大学工学部情報科学科
Faculty of Engineering, Okayama University

ェアリソースを，他方の OS に提供する方式について提案し，PCI ハードウェアに適用する．

２．MRM（複数実計算機）

従来の仮想計算機方式と比較した MRM の特徴は，仮想計算機方式が一つの OS のメモリ空間上で他方の OS が走行することに対し，MRM では，二つの OS が独立に走行する点である．

また MRM では，タイマ，CPU などの共有必須であるリソース以外は，各 OS がリソースを占有することで，各 OS の性能を向上させる．そして，他の OS が占有しているリソースからの割込みを契機に，走行する OS を切替える．

３．非共有リソース利用方式

OS は，基本的に I/O 命令を発行することによって，ハードウェアリソースを制御する．そこで，あるハードウェアを占有して制御する OS は，そのハードウェアに対応する I/O 命令を適切に発行する必要がある．一方で，同時走行するそれ以外の OS，つまりそのハードウェアを制御することを許されていない OS は I/O 命令を発行してはならない．つまり，ハードウェアを占有して制御する側の OS の内部状態と実際のハードウェアの状態との間の一貫性を保つために，一方の OS が独占的に I/O 命令を発行することが必要となる．従って，二つの OS 間でハードウェアリソースを共有する場合，一方の OS が発行した I/O 命令を，本来，そのハードウェアを占有すべき OS が横取りして，ハードウェア制御の調停を行う必要がある．そうすることでハードウェア制御の一貫性を保ち，他方の OS からはあたかも自身がハードウェアを占有して I/O 命令を発行し，ハードウェアを制御しているかのように振舞うことができる．

図１に，本方式の概念を示す．

図１において，機能提供を受ける側である OS1 から発行される I/O 命令は，OS2 によって，以下三点のいずれかの処理が実施される．

(1) OS1 が占有するハードウェアへの I/O 命令であれば I/O 命令を実行させる．
(2) OS2 のハードウェアであり，リソースを共有しないハードウェアであれば，I/O 命令を実行させず，エラー値を返す．

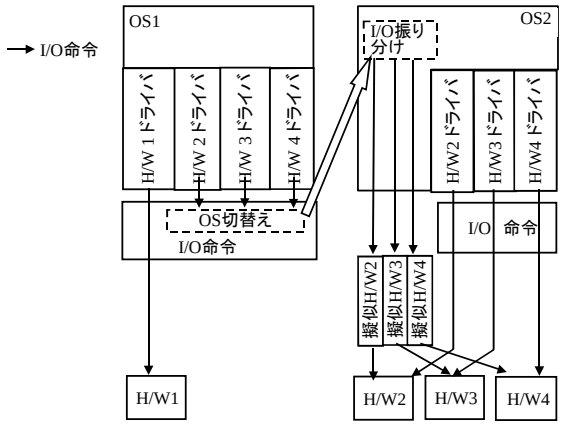


図１非共有ハードウェアのリソース利用方式．

(3) OS2 のハードウェアであり，リソースを共有するハードウェアについては，擬似 H/W に処理を依頼する．

図１では，H/W1 を OS1 が占有し，残りを OS2 が占有している．このとき，OS1 からの I/O 命令は，OS を切替える処理を通じて，OS2 に伝えられる．

OS2 は，OS1 側が発行する I/O 命令の I/O アドレスと，ハードウェアのレジスタとの対応表を持ち，上記 (1) ～ (3) までのいずれかの処理を行う．

図１では，H/W2，H/W3 についてはリソースを共有するため，H/W2 ドライバ，H/W3 ドライバからの I/O 命令は，OS 切替え処理を通じ，OS2 に伝えられる．その後，ハードウェア制御の調停の役割を果たす各擬似 H/W が I/O 命令を処理する．H/W4 はリソースを共有しないため，擬似 H/W に I/O 命令は伝わらず，エラーとなる．

擬似 H/W が I/O 命令を処理について，OS1 のドライバからの命令を全て実行すると，OS2 のドライバが H/W に対して行う制御を妨害する可能性がある．そこで擬似 H/W は，できる限り I/O 命令を自身で処理するものとし，H/W の機能を必要とする命令については，OS2 のドライバが H/W を制御する処理を妨害しないように I/O 命令を実行する．

４．非共有リソース利用方式の PCI ハードウェアへの適用

PCI ハードウェアは，NIC，ビデオカード，RAID コントローラなど，現在様々なハードウェアが対象となる主流なハードウェア間の通信規格である．

そこで，PCI ハードウェアに本方式を適用することで，本方式が様々なハードウェアに適用できることを述べる．なお，図 1 において，OS を切替える処理は既に述べた[2]．そこで，以降は簡単のために，OS を切替える処理を省略して考える．このとき，OS2，OS2 の H/W1 ドライバ～H/W4 ドライバは考えなくともよく，図 2 を考えればよい．

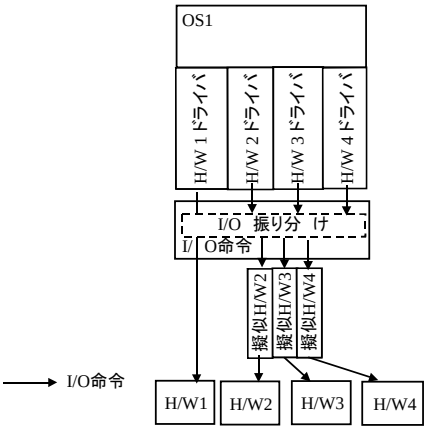


図 2 単独 OS における非共有リソース利用方式

4.1 要求条件

3 章で述べたように，本方式では，I/O 命令を各擬似 H/W に振分けることが必要である．このとき，各ハードウェアのレジスタと，I/O アドレスの対応を知っておく必要がある．

図 2 では I/O 命令が，I/O アドレスと各ハードウェアのレジスタとの対応を把握する．また，図 1 で OS2 が行っていた I/O 命令の振分けを行う．

I/O アドレスとレジスタとの対応を取る方法は，ハードウェアの種類により異なる．すなわち，対応付けには，I/O アドレスとレジスタが常に同じ I/O アドレスに対応付けられるハードウェアと，OS が対応付ける I/O アドレスを設定可能なハードウェアがある．

このうち，前者であれば，図 2 の I/O 命令が，静的に対応情報を持てばよい．しかしながら，後者のときは，OS が I/O アドレスとレジスタを対応させる状況に応じて，異なる対応情報を持つ必要がある．

PCI ハードウェアでは，I/O アドレスとハードウェアのレジスタの対応付けは，OS が設定する．このため，図 2 の I/O 命令は，OS が I/O アドレスと各ハードウェアのレジスタを対応付ける設定を行う際に，動的に対応情報を生成する必要がある．

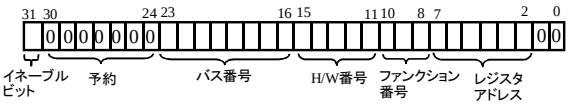
4.2 PCI コンフィグレーション空間

PCI ハードウェアでは，ハードウェアが持つレジスタのうち，PCI ハードウェア共通の設定を行うレジスタを常に同じ I/O アドレスに対応付ける．その他のレジスタについては，OS が対応付ける I/O アドレスを設定可能としている．このうち，PCI 共通のレジスタが対応付けられるアドレス空間を PCI コンフィグレーション空間と呼ぶ．

PCI コンフィグレーション空間に対応付けられるレジスタに対して I/O を行う方法は，その他のレジスタに I/O を行う方法とは異なり，以下（1），（2）の順に，図 3 のフォーマットにおいて I/O を行うことである．

（1）I/O アドレス 0xCF8（以降，PCI アドレスポート）にダブルワードの OUT 命令を行い，I/O を行うハードウェア，レジスタを指定する．
（2）I/O アドレス 0xCFC～0xCFF（以降，PCI データポート）に I/O を行うことで，（a）で指定したレジスタに I/O を行うことができる．

（1）PCIアドレスポート（0xCF8）の出力フォーマット



イネーブルビット　：PCIコンフィグレーション空間へのI/O（1のとき）
　　上記以外のI/O（0のとき）
バス番号、H/W番号　：H/Wが存在するバスとスロットの番号
ファンクション番号　：H/Wが多機能である場合には、コンフィグレーション空間を複数持つことが想定される
レジスタアドレス　：コンフィグレーション空間内のレジスタを指定

（2）PCIデータポート（0xCFC～0xCFF）のフォーマット

PCIアドレスポートで出力したレジスタ番号を4で割った余りによってI/Oアドレスが変化

余り	I/Oアドレス
0	0xCFC
1	0xCFD
2	0xCFE
3	0xCFF

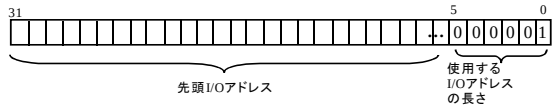
図 3 PCI コンフィグレーション空間へ I/O を行う方法

PCI コンフィグレーション空間に対応付けられるレジスタは複数あり，I/O を行うレジスタは，図 3 の PCI アドレスポートのうち，レジスタアドレスで指定する．

4.3 レジスタに対応付ける I/O アドレスの設定方法

ベースアドレスレジスタは，PCI コンフィグレーション空間に対応付けられるレジスタの中の一つである．OS は，ベースアドレスレジスタに，I/O アドレスを書込むことで，PCI コンフィグレーション空間に対応付けないレジスタを任意の I/O アドレスに対応付ける．

ベースアドレスレジスタのフォーマットを図 4 に示す．ベースアドレスレジスタには，I/O アドレスが対応付けられるレジスタ数に応じて，読出し専用ビットが設置される．図 4 の場合は 6 ビットであり，2^6 ＝ 64 の I/O アドレスを使用する．



※ 使用するI/Oアドレスの長さは，2ビット目から続く
読み出し専用のビットの長さで表す
上図の場合は0～5ビットのため、64が長さとなる

（ ただし，先頭I/Oアドレスも指定するため，16ビット目まで）

図 4 ベースアドレスレジスタのフォーマット

以上述べた事項より，各 PCI ハードウェアのベースアドレスレジスタに対する I/O は，4.2 節（1），（2）で述べる方法を，図 3，図 4 に述べるフォーマット用いて，レジスタアドレスを 0x10 として実施すればよい．なお，図 3 のバス番号，ハードウェア番号，ハードウェアが設置される場所による．

また，各ハードウェアのレジスタが I/O アドレスに対応付けられる手順は 4.2 節の（1），（2）で述べる方法を用いて，以下（A）～（B）となる，

（A）ベースアドレスレジスタに 32 ビット全て 1 の値を書き出す．
（B）ベースアドレスレジスタの値を読込む．
（C）ベースアドレスレジスタに，レジスタを対応付けたい先頭 I/O アドレスを書込む．

OS は，上記（A）～（C）により，ハードウェアが使用する I/O アドレス数を知ることができる．これを用いて，図 4 に示す通り，対応付ける先頭 I/O アドレスを書込み始めるビット数がわかるため，上記（c）により，レジスタを I/O アドレスに対応付けることができる．

4.4 ベースアドレスレジスタに対する設定内容の把握

4.1 節で述べたように，本方式では，I/O 命令の I/O アドレスと I/O 値から，I/O 命令が実行されるハードウェアのレジスタを特定する必要がある．また，本方式を PCI ハードウェアに適用するためには，OS が各 PCI ハードウェアのレジスタと I/O アドレスを対応付けるよう設定するときに，設定内容を把握する必要がある．

そこで，本方式では図 2 の I/O 命令内で，図 5 に示す処理を行い，設定内容を把握する．

図 5 では，入力として，I/O 命令を各 H/W のドライバから受け取る．次に，4.2 節で述べたように，PCI コンフィグレーション空間に関する I/O は，他と比べて，I/O を行う方法が異なるため，I/O 命令判断前処理において PCI コンフィグレーション空間への I/O とそれ以外の I/O を分ける．分ける判断基準は，図 3 で示した PCI アドレスポート，PCI データポートに合致する I/O アドレスであることである．

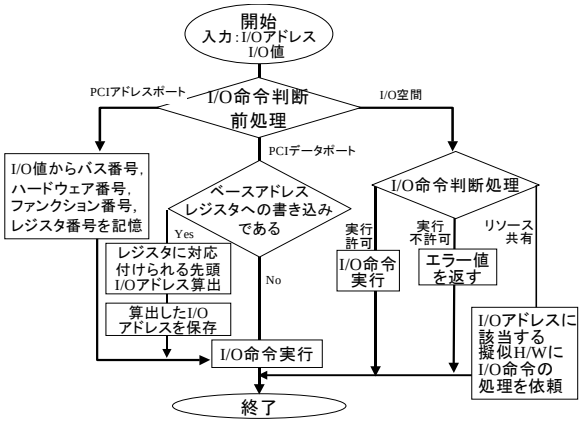


図 5 I/O 命令内の処理

分類後は，PCI コンフィグレーション空間への I/O ならば，I/O 命令を実行する．ただし，ベースアドレスレジスタへの I/O については，ハードウェアのレジスタと I/O アドレスを対応付ける命令であるため，対応付ける I/O アドレスの内容を含む I/O 値を把握したい．そのため，全ての PCI アドレスポートへの I/O 命令の処理として，I/O 値からバス番号などを取得した後，I/O 命令を実行する．図 6 に PCI データポートへの I/O の詳細処理フローを示す．

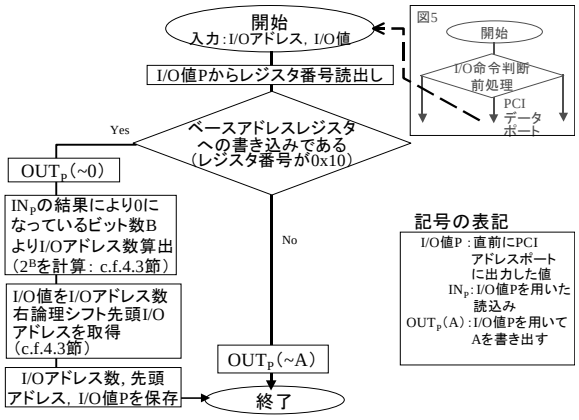


図 6 PCI データポートの際行う処理

図 6 により任意のバス番号，ハードウェア番号，ファンクション番号を持つハードウェアのレジスタに対応付けられた I/O アドレスを把握できる．

また，PCI コンフィグレーション空間への I/O 命令以外であれば，3 章の（1）～（3）で述べる I/O 命令を実行させる／エラー値を返す／擬似 H/W に処理を依頼する，のうちいずれかを実行する．

4.5 擬似 H/W の選定方法

4.4 節において，各ハードウェアがレジスタと対応付ける I/O アドレスを得た．しかしながら，リソースを共有する場合には，I/O 命令の処理を依頼する擬似 H/W も決定する必要がある．この処理は，図 5 の I/O 命令判断処理で行われるため，この処理の詳細について述べる．

I/O 命令判断処理とは，図 5 に示すように入力である I/O アドレスを使用するハードウェアを特定し，事前に設定しておいた 3 章の（1）～（3）の事項に基づいて，該当する擬似 H/W に処理を依頼するなどの処理をすることである．また，ユーザがリソースを共有しないとしたハードウェアは I/O 命令を実行させず，このため，I/O 命令判断処理には，以下の情報が必要である．

（1）各ハードウェアレジスタの I/O アドレス
（2）ハードウェアのリソース共有に関する情報（I/O 命令を実行／I/O 命令を実行させない／リソース共有するのいずれか）
（3）（リソース共有をする際には）該当する擬似ハードウェア
（4）各ハードウェアが設置されるバス番号，ハードウェア番号，ファンクション番号

（4）について，3 章の（1）～（3）に述べたリソース共有するなどの事項は，ユーザが決定する．これらの事項を設定する際に，まず設定するハードウェアを決定する．このとき，ユーザと OS の双方が認識できる情報が必要であり，PCI の場合は上記（4）が適当である．

図 7 に I/O 命令判断処理の処理内容と，ユーザの初期設定との関係を示し，説明する．

ユーザには，初期設定ファイル（SG 表：system generation 表）を用意させ，OS ブート時に，設定した情報を初期設定ファイルから読出す形式で管理する．この設定情報の管理を行うデータ構造を，以降管理表と呼ぶ．なお，SG 表に擬似 H/W のアドレスが追加された理由は，擬似 H/W を OS に組み込むタイミングは，ユーザが決定するからである．

図 7 に，SG 表と管理表に含まれる項目と，図 5 の I/O 命令判断処理の詳細フローを示す．

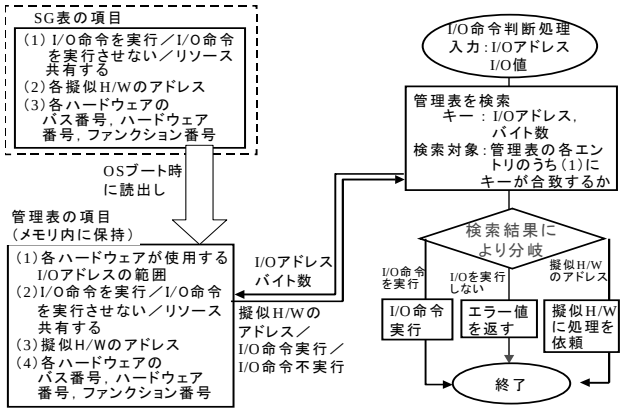


図 7 管理表と SG 表の役割

I/O 命令判断処理は，I/O アドレス，バイト数をキーに管理表を検索し，その戻り値を元に，次の処理を決定する．なお，ハードウェアによっては，レジスタに I/O を行う際に，バイト数によって処理が変わる場合があるため，一つの I/O アドレスによって複数の処理が考えられ，そのため，バイト数もキーとなっている．

5．非共有リソース利用方式の実装

5.1 実装方針

4 章で述べた方式の拡張を実装するにあたり，以下二つの方針で実装する．

（方針 1）パフォーマンスの劣化を防ぐため，検索効率を下げない処理を優先する．また管理表のデータ構造が占めるメモリ領域を抑える．
（方針 2）あるハードウェアについて，容易にリソース共有と非共有を切替えられるようにする．例えばあるハードウェアについてリソース共有する際に，大幅に OS のソースコードを変更しなければならないとしたら，利便性を損なうためである．

5.2 管理表と擬似 H/W のデータ構造

管理表と擬似 H/W の構成を図 8 に示し，説明する．

まず，管理表の m 番目が I/O アドレス 4m～4m+3 用のエントリを示すとした（m： 0 ≦ m ≦ 0x3fff）．

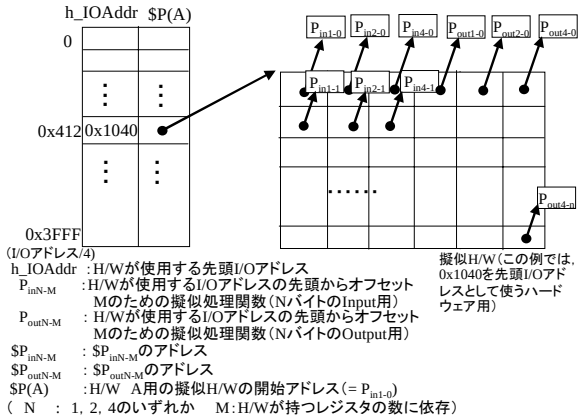


図 8 管理表と擬似 H/W のデータ構造

すなわち，1番目のエントリは 0x00～0x03 までの I/O アドレス用である．また，各エントリは，I/O アドレス 4 つを管理するが，この I/O アドレスを使用するハードウェアの先頭 I/O アドレスを保持する．このため，入力となる I/O アドレスを 4 で割った商から，管理表で参照する箇所が求められる．

管理表の項目は，各エントリを使用するハードウェアの先頭 I/O アドレスと，そのハードウェア用の擬似 H/W の処理を行うプログラムの開始アドレスで構成される．

なお，先頭 I/O アドレスに関しては，例えばこの要素の値が 0 ならば I/O 命令を実行，1 ならば I/O 命令を実行させないものとし，それ以外のものはリソース共有とする意味も含む．

また，擬似 H/W では，使用する I/O アドレスの先頭から 0 バイト離れた箇所の擬似化処理ルーチンへのポインタについて，1 バイトの Input 用処理，2 バイトの Input 用処理…4 バイトの Output 用処理のように並べ，次に I/O アドレスの先頭から 1 バイト離れた箇所の擬似化処理について，1 バイトの Input 用処理…のように並べる．このように構成した擬似 H/W を，各ハードウェアごとに用意する．

この構造を用いると，入力 I/O アドレスに該当する擬似 H/W 内の擬似化処理ルーチンへのポインタを格納するインデックスは，擬似 H/W の先頭から以下の演算の値だけ加えたものとなる．

（入力 I/O アドレス－管理表から求められる先頭 I/O アドレス）×6＋α

実際には，アドレスを示す値は 4 バイトであることを考え，上記の値を 4 倍する必要がある．なお，α とは，I/O 命令の並び順であり，例えば 1 バイトの Input 命令なら α=1，2 バイトの Input 命令なら α=2…4 バイトの Output 命令なら α=6 となる．

図 8 を用いて，実際に擬似 H/W の処理を実行するまでを，I/O アドレス 0x1049 に対する 2 バイトの読込みを例として，図 9 に示す．

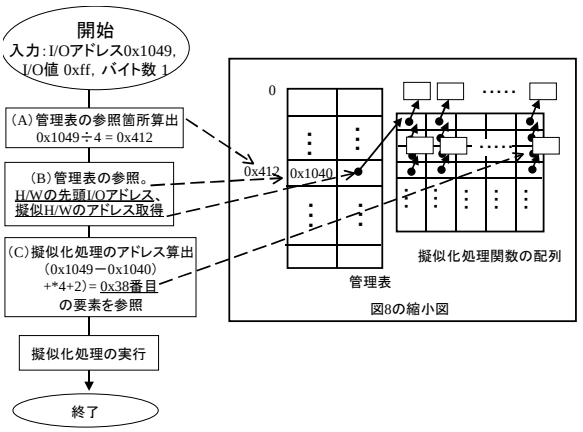


図 9 管理表から擬似化処理を実行するまでの例

管理表を構成する上で，m 番目のエントリを I/O アドレス m 用とすることも考えられる．この方法では，表のサイズが増大するため，I/O アドレス 4 単位で 1 エントリを構成した．4 単位とした理由は，I/O アドレスは 4 バイト単位でハードウェアのレジスタに対応付けられるため，各エントリで，複数の擬似 H/W へのアドレスを持つことを避けることができるためである．

また，新規にリソースを共有するとき，必要となる事項は，擬似 H/W の組込みと，SG 表の記入である．このため，擬似 H/W の組込みを容易であり，かつ管理表の大幅な修正を伴わないように管理表を構成したい．すなわち，擬似 H/W を OS に組み込むだけで，管理表に変更を加えず，SG 表の内容を修正するだけとしたい．そこで，あらかじめ管理表に各エントリを使用するハードウェアの先頭 I/O アドレス，擬似 H/W の先頭アドレスを加える．

次に，できるだけ表のサイズを縮小させ，パフォーマンスの劣化を防ぐために，I/O 命令を実行／I/O 命令を不実行／リソース共有を示す項目は，擬似 H/W の先頭アドレスを示す項目に統合した．すなわち，例えばこの要素の値が 0 ならば I/O 命令を実行，1 ならば I/O 命令を実行させないものとし，それ以外のものはリソース共有とみなす．

また，各ハードウェアのバス番号など，必要の

ないデータ構造は，表のサイズを縮小するために管理表には加えない．SG 表からメモリ上に読出し，4.5 節 (1) で述べるハードウェアレジスタの I/O アドレスが OS ブート時に挿入された時点で破棄するものとした．

5.3 SG 表のデータ構造

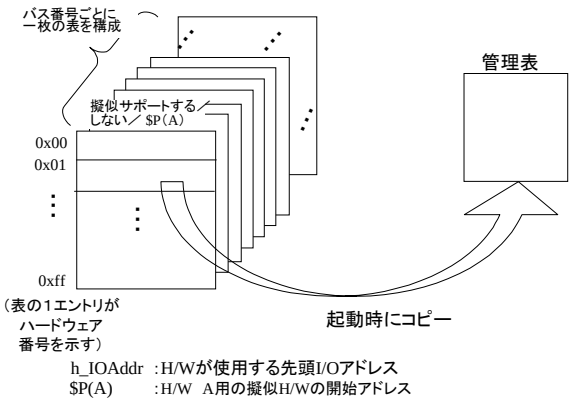


図 10 SG 表のデータ構造

SG 表に関して，方針 1，方針 2 の中で留意すべき事項は，表のサイズが増大し，OS ブート時のパフォーマンスの妨げとならないようにすることである．図 10 に，図 7 の SG 表を実装した様子を示し，以下に説明する．

(1) 図 7 の SG 表における，I/O 命令を実行する／I/O 命令を実行させない／リソース共有する項目と，擬似 H/W の処理を行うプログラムの開始アドレスは，表のサイズを縮小するために図 8 と同様に統合する．
(2) ハードウェアのバス番号，ハードウェア番号，表のサイズを縮小するために，バス番号ごとに SG 表を用意する．また，表の各エントリが，ハードウェア番号を示すように表を構成する．

6．複数実計算機に適用する際の変更点

本稿では，簡単のために OS を一つという前提で述べた．これを我々が提案した MRM に適用するためには，以下二点を考慮に入れる[1].

(1) 管理表は OS2 が作成し，OS2 が保持する．
MRM では，一方の OS（OS2）が他方の OS（OS1）を監視しながら起動させる[3]．OS は，起動時にハードウェアを認識するために I/O 命令を行う．このとき，OS2 のハードウェアを OS1 に認識させないために，OS2 が管理表を保持する．
(2) 管理表／SG 表の追加項目
本方式では，図 2 より，I/O 命令は全て OS2 が一度受け取る形式である．そのため，OS2 は，OS1 のハードウェアに対する I/O 命令であれば，I/O 命令を実行する．このとき，OS1／OS2 のハードウェアであることを示す項目が必要になる．

また，課題として，I/O 空間の衝突がある．
複数の OS が走行する環境下では，各 OS が独自の方法で I/O 空間にハードウェアをマッピングする．そのため，ある I/O アドレスが，OS1 では NIC のレジスタにマッピングされているが，OS2 では他のハードウェア用にマッピングされることも考えられる．この課題は，以下 (A) ～ (C) を実施することで対処する．

(A) 一回目の起動時には，OS2 がハードウェアを使用しないものとして，OS1 を一度 OS2 の監視下の元で起動させる．このとき，OS2 は，OS1 が，I/O アドレスと各ハードウェアのレジスタを対応付ける情報を取得可能である．その後，OS1 を停止させると OS2 はハードウェアが使用可能となるため，取得情報を SG 表として保存する．
(B) 二回目に起動する前に，SG 表を元に，OS2 が，OS1 のハードウェアを I/O 空間にマッピングする状況に合わせる．
(C) 二回目以降は，OS2 も OS2 用のハードウェアを使用するとして起動を行う．

7．本方式の応用例

本方式で用いた I/O 命令のフックと，擬似 H/W には，様々な応用が考えられる．ここでは，その応用例について述べる．

(1) ハードウェア利用履歴の取得
I/O 命令をフックすることにより，ハードウェアの利用履歴を取ることができる．この応用例として，以下二つが応用例として考えられる．

(1-a) ハードウェア不正利用の防止

近年では，悪意のある第三者が，カーネルモジュールとしてバックドアを仕掛ける攻撃がある．このような攻撃では，例えばハードウェアに直接 I/O を行われ，OS がハードウェアを使用する処理を妨害することが考えられる．これを防止する対策は，アプリケーションレベルでは困難であるだけに，OS レベルで I/O 命令をフックする防止策は有効である．

(1-b) OS やドライバのデバック

OS 内では，様々なドライバを用いて，ハードウェアを制御する．例えば新規 OS を開発する際には，各ドライバについて入念なデバックを行った上でも，潜在的なバグは考えられる．このとき，本方式で用いた I/O 命令のフックを用いて，多数のドライバに対する I/O の履歴を一元的に管理して，デバックに使用できる．

(2) パケットフィルタリングの性能向上

一般にパケットフィルタリングは，ファイアウォールを用いて，カーネル内で行う．他方で，擬似 H/W を用いると，より下位の層でパケットフィルタリングを行うことになり，通信を許可しないパケットを早期に落とすことができる．このため，余分なパケットに対する処理が削減可能となり，パケットフィルタリングに関する性能向上が期待できる．このときの利点とは，例えば計算機のサービス不能攻撃（Denial of service attack）に対する耐性を高める手段となることが考えられる．

以下に，一般に考えられる通信の受信処理を示し，擬似 H/W を用いたフィルタリングと，通常のファイアウォールとの比較を行う．また，図 11 に両者を比較する図を示す．

(A) NIC が，NIC 内のバッファから，DMA 転送を用いて，ドライバのバッファにパケットをコピーし，割込みを入れる．

(B) ドライバが，各パケットについて DMA 転送時のエラーチェックなどを行った後，カーネル内のバッファにパケットをコピーする．

(C) カーネル内で，各プロトコルごとの処理（チェックサムの計算など）を行う．

(D) パケットフィルタリングを行った後，ユーザ空間にパケットをコピーする．

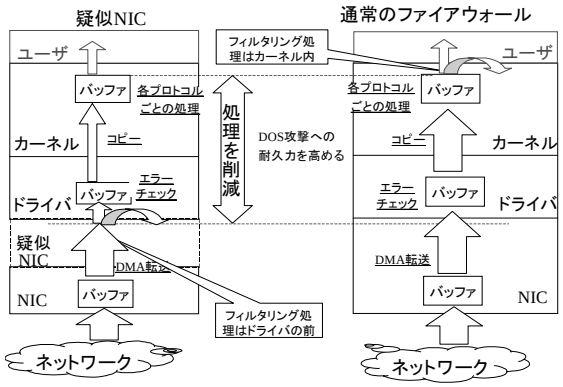


図 11 擬似 H/W と I/O のフックを用いた OS

擬似 NIC は，NIC ドライバと NIC の間に存在する．このため，通常 NIC からドライバに入る割込みを，擬似 NIC にフックさせることが可能となる．そのため，割込みをフックする際に，パケットフィルタリングを実施可能となり，遮断するパケットに関する上記 (B)，(C) の処理を削減できる．

8. おわりに

ここでは，我々が提案した複数実計算機環境における，非共有ハードウェアの利用方式を述べ，本方式を PCI ハードウェアに適用した．本方式は，I/O 命令のフックを行うことで，複数実計算機環境におけるハードウェア制御の一貫性を保ち，また擬似 H/W によって，ハードウェアを占有しない OS にもハードウェアリソースを提供する方式である．

今後は，本方式を複数実計算機に対して実装することが挙げられる．

参考文献

[1] 谷口秀夫，乃村能成，田中一男，大塚作一，井上友二，“ハードウェアを非共有する複数オペレーティングシステムの構成法，”情報処理学会研究報告，Vol. 2002, No. 79, pp. 47-54, 2002.

[2] 桝本圭，中島雄作，伊藤健一，乃村能成，谷口秀夫，“複数 OS 環境における OS 切替え方式，”情報処理学会研究報告，Vol. 2003, No. 19, pp. 23-30, 2003

[3] 田渕正樹，中島雄作，伊藤健一，乃村能成，谷口秀夫，“二つの OS 共存に向けた起動方式，”情報処理学会研究報告，Vol. 2003, No. 19, pp. 15-22, 2003