PCIデバイスを監視する統合的な処理機構の提案

乃村能成[†] 山本裕馬[†] 谷口秀夫[†]
桝本圭[††] 伊藤健一[††]

従来，オペレーティングシステム（OS）とハードウェアアーキテクチャとの関係は，多様なアーキテクチャの差異をOSが吸収するというものであった．これにより，OSはアプリケーションに対して同様のソフトウェア環境を提供していた．しかし近年では，PCアーキテクチャ上で多様なOSを同時に動作させ，使用するOSを逐次変更する逆の関係も現れた．また，OSとハードウェア間に，複数のOSが共有する機構を設けることにより，複数のOS間でハードウェアを共有するという考えも生まれた．我々は，この考え方に注目し，従来デバイスドライバ上で行われていたPCIデバイスの状態監視をOSとハードウェアの間で行うための処理機構を提案し，同時に走行する2つのOSからハードウェアを共有した例を示す．本提案は，低レベルで監視を行うことにより，詳細かつ効率の良い監視を可能とする．

A Method for Monitoring of PCI Devices

Yoshinari Nomura,[†] Yu-ma Yamamoto,[†] Hideo Taniguchi,[†]
Kei Masumoto[††] and Ken-ichi Itoh[††]

Operating systems had been supposed to provide a single environment to their applications concealing details of various hardware architectures. However, in recent years, we can observe a different relationship between operating systems and hardwares: users run various operating systems concurrently or alternatively on a single PC. And more, these operating systems are able to share hardware components of a PC using a kind of hardware sharing mechanism. Based on this mechanism, we propose a program design framework to monitor PCI devices. Such kind of softwares were conventionally embedded in device drivers, on the other hand, our framework is on off-the-OS level, and has a lower level inspection method which brings us an effective and precise monitoring.

1. はじめに

オペレーティングシステム（以下OS）の目的の1つは，アプリケーションプログラム（以下AP）にハードウェアを直接意識させないようにすることである．つまり，異なるハードウェアアーキテクチャ上においても，同一のソフトウェア環境を提供するということが大きな目的とされてきた[1)]．

そのため，OSを複数のハードウェアアーキテクチャに対応させるためのデバイス抽象化やソフトウェアの移植性向上についての議論が多くなされている．

一方で，近年，パーソナルコンピュータ（以下PC）のアーキテクチャは，x86とPCIバスという構成が大半を占めるに至っており，それを前提として動作するAPが多く存在している．たとえば，マルチメディアアプリケーションが画像操作のための特殊なハードウェアデバイスを操作する場合，OSが提供するデバイス抽象化が不十分であるために，そのハードウェアデバイスの機能や性能を十分に生かすことができないことがある．そのような場合，OSはハードウェアを直接APに見せるための窓口を用意するだけの構造を備えて，実際はハードウェアに直結したプログラム構造をAPに要求することがある．そのような場合，APは，PCアーキテクチャに強く依存しているが，OSにはさほど依存しないといった構造を持っている．

また，PCアーキテクチャを1つのデファクトとして，その上で複数のOSを動作させるという，これまでとは違った要求が出てきている[2),3)]．つまり，多種のハードウェアに対して単一のOSを提供するのではなく，ハードウェアは共通で，OSをユーザの嗜好や目的に応じて変更するという，これまでのOSとハー

† 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
†† 株式会社NTTデータ
NTT DATA Co.

ドウェアの関係を考えるうえで指向されてきたものとは逆の構造を指向している．

このような状況で x86 CPU アーキテクチャに特化した CPU の利用を OS が指向したり，PCI バスという共通のアーキテクチャ上で，複数の OS が PCI 資源をシェアしたりするといった考え方が生まれてくる[4]．

この考え方に基づいて，これまでの OS がハードウェアデバイスの状態監視をするために，デバイスドライバに一般的な統合メソッドを用意し，OS が提供するといった形態に加えて，OS の外部に PCI バス上のデバイスをじかに操作して，デバイスに特化された，（ただし OS には特化されない）情報を取り出し，監視する部分を用意することを考える．

本論文では，これらの観点から PCI デバイスを監視する統合的処理機構を提案する．これによって OS に依存しない形でデバイスを監視したり，PCI デバイスの動作を動的に変えたりする方法を述べる．これをたとえば，NIC のような PCI デバイスに応用した場合，パケット監視や，フィルタリングを OS の介在なしに行うことができることを意味する．

デバイスの監視に関する技術として，XenoServer[5] や VMWare[6]，また Denali[7] のように，仮想計算機技術を用いて複数の OS を同時に稼動させる技術がある．これらは，1 つの主となる OS の上で，他の OS をアプリケーションのように稼動させ，他の OS がデバイスを制御する処理を主となる OS で監視可能とする．

また，U-Net[8),9)] のように，通信を高速化するために各 AP がユーザ空間に設置された仮想ネットワークデバイスを用いて，カーネル空間を経由せずに通信を行う技術がある．Ensamble[10)] は，この仮想ネットワークデバイスを監視して，通信制御や QoS を実現する．

前者の研究は，複数の OS が稼動する環境にも対応可能であり，後者の研究は，各 AP に近い監視を可能とする．一方，本機構は，OS の外部に監視部分を用意するという点で，前者と同様の構造を持たせることも可能であるが，監視側がすべてのハードウェアを仮想化して監視対象に見せる方式とは異なり，監視対象のハードウェアのみに特化して監視処理を挿入することができる．そのため，より効率的に低レベルの監視を実現することが可能になると考える．

2. 目　　的

本機構と，OS，デバイスの関係を**図 1** に示し，本機構の位置付けを説明する．OS はデバイスドライバを用いてレジスタに対して I/O 命令を発行し，制御を

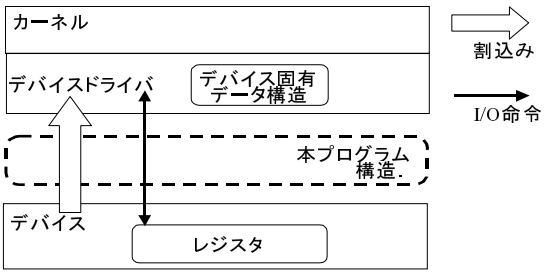

図 1 本機構と OS，デバイスの関係
Fig. 1 Relation between OS, device and our method.

行う．ここでは，I/O 命令とは I/O 空間に対する I/O のことであるとし，メモリ・マップド I/O は対象外であるとする．I/O 命令によりデバイスは，内部状態を遷移させ，デバイスの機能を OS に提供する．また，機能を提供する過程で，内部状態を OS に通知するために割込みを発生させる．また，デバイスは，DMA を用いたデータ転送のように，OS 側の領域を利用して機能提供することもある．このとき OS は，デバイスから指定された形式のデータ領域を用意し，I/O 命令を用いて領域の情報を事前にデバイスに伝える．

この関係において，本機構は OS と図 1 の点線部でデバイス制御の監視を行い，I/O 命令，割込みの発生，用意したデータ構造を監視する．このように，OS の外部かつデバイスに近い部分でデバイス監視を行うことで，以下の利点が生じる．

(1) 監視処理を追加する際の OS の変更が少ない．
OS がデバイスドライバに一般的な統合監視メソッドを用意する従来の形式では，新しい処理の追加は OS の変更をともなう．それに対し本機構は，OS とのインタフェースは追加するが，OS への変更を少なくとどめられる．

(2) 監視処理の移植性が高い．
デバイスに近い，低いレベルで監視を行うため，OS に依存しない監視を行うことができる．すなわち，本機構を用いた監視処理を容易に他の OS に適用することが可能である．

(3) 複数の OS に同時に対応ができる．
デバイスに近い，低いレベルで監視を行うため，複数の OS が同時にデバイスの制御を試みる環境下にも対応が可能である．

(4) 詳細かつ効率の良い監視が可能である．
デバイスに対する 1 処理ごとに監視を行うことで，詳細な監視が可能である．また，本機構は，OS とデバイスの間に位置するため，各制御処理が実行される前に監視を実施することや，デバイスドライバまたは OS 内部にあるデバイス

固有のデータ構造をデバイスの状態遷移が発生する前に監視することで，効率の良い監視が可能である．

このように，本機構は，OS 内部の構造とは極力切り離されているために多くの利点を持っている．しかしながら，OS 内部の状態を監視することはできないため，OS が管理している情報（たとえばシステムのメモリ使用量等）とデバイスの状態を関連付けて監視することはできない．

3. PCI デバイスを監視する処理構造

3.1 基 本 構 造

PCI デバイスを監視する基本構造を**図 2** に示し，以下に処理の概要を説明する．

(1) PCI 監視処理構造の挿入
あらかじめユーザが監視対象のデバイスを指定したうえで，本構造を監視対象の OS からは見えない形で配置する．つまり全物理メモリのうち，OS が管理する部分から一部を本構造のために予約し，別の仮想空間上に配置する．

(2) I/O 命令の横取り処理
OS が発行する I/O 命令を横取りして，仮想空間を切り替えて，本構造の I/O 命令解析処理部に処理を移す．

(3) 割込み処理の横取り処理
デバイスからの割込みを横取りして，仮想空間を切り替えて，本構造の割込みの監視処理部に処理を移す．

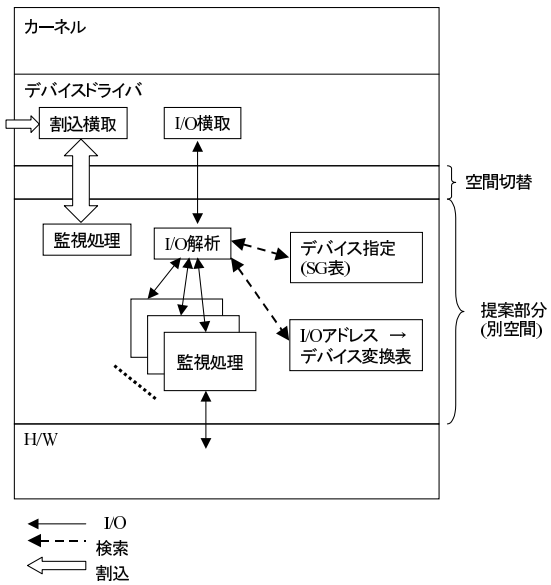

図 2 PCI デバイスを監視する基本構造
Fig. 2 Basic structure.

(4) I/O 命令の解析処理
横取りした I/O 命令を解析し，I/O 命令が監視対象デバイスに対する操作かどうかを判断する．監視対象デバイスに対する I/O 命令だと判断した場合は，実際の監視処理を起動する．

(5) 実際の監視処理
発行された I/O 命令に基づいて，デバイスに対する操作を監視する．また，監視処理の結果に基づいて，本来 OS が発行した I/O 命令をそのまま，あるいは変更してデバイスに発行する．

これらの処理を実現するうえで，以下の課題が考えられる．

(課題 1) OS が実行する I/O 命令の横取り方法
OS が発行する I/O 命令を横取りして，本構造に制御を移す仕組みが必要である．

(課題 2) デバイスが発生させる割込みの監視
デバイスを監視するためには，I/O 命令だけでなく，デバイスが OS に対して内部状態を通知するための割込みの発生も把握する必要がある．

(課題 3) PCI デバイスと I/O アドレスの対応把握
監視対象のデバイスに対する I/O 命令を把握するためには，I/O 命令のアドレスとデバイスとの関係を判断する必要がある．しかし，I/O アドレスが対応するデバイスは OS 初期化時の状況で異なるため，状況に応じて，I/O アドレスとデバイスの対応関係を動的に把握する必要がある．

(課題 4) 監視対象 PCI デバイスの指定方法
各デバイスの監視有無をユーザが指定する方法が必要である．この際，監視対象となるデバイスは多岐にわたるため，すべての PCI デバイスについて一律に適用できる方法が必要である．一方，ユーザは I/O アドレスを事前に把握することはできない．そのため，具体的には，バススロットの何番目に装着しているデバイスを対象とするといった指定方法が望ましい．

(課題 5) 監視の実施有無の効率的な決定
監視処理実行の契機は，I/O 命令または割込みの発生である．これらの契機と，課題 3 で述べたデバイスと I/O アドレスの対応関係，さらに課題 4 で述べた監視有無の情報を組み合わせることで，監視処理の実施有無を決定する方法が必要である．このときのオーバヘッドは，デバイスを使用するうえでのパフォーマンスに大きく影響するため，高速な処理を必要とする．

(課題 6) 監視処理の粒度の細かい実行

本構造では，OS とデバイスの間に存在する利点を生かし，細かい監視内容を規定可能としたい．たとえば，I/O デバイスには，メモリと違い，書き込みと読み出しで違う意味を持っていることがある．あるいは，バイト単位とワード単位のアクセスでは挙動を変えるデバイスもある．そこで，1 つのアドレスに対する I/O 命令でも，複数の処理を規定することになるが，そのときでも該当する監視処理を高速に抽出し，デバイスのパフォーマンスの低下を防ぐことが必要である．

（課題 7） 本構造の挿入方法

本構造をできるだけ監視対象 OS に意識させることなく挿入する方法を検討する必要がある．

課題 1，2，7 は，OS の実行コードをソースコードレベルで書き換えたり，動的に実行バイナリを変更したりすることによって実現する．つまり OS が意識する必要がある部分である．しかしながら，そのための OS の変更は，最小限であることが望ましい．一方，課題 3〜6 は，本構造の中で実現すべき課題である．

以降，4 章では，議論の前提となる PCI デバイス制御の基本構造について説明し，5 章ではこれらの課題について対処を述べる．

4. PCI デバイス制御の基本構造

4.1 I/O アドレス空間

OS は I/O アドレス空間内に I/O 値を読み書きすることで，デバイスを制御する．なお，I/O アドレス空間は，16 ビット幅（0〜0xFFFF）である．

図 3 に I/O アドレス空間を示す．I/O アドレス空間は，全 PCI デバイスを管理する領域と，各デバイス（PCI もそれ以外のデバイスも含む）を制御する領域の 2 つに分類できる．前者の領域は，0xCF8〜0xCFF の部分である．前半の 4 バイト（0xCF8〜0xCFB）は，書き込み専用の部分であり，後半の 4 バイト（0xCFC〜0xCFF）で読み書きするデバイスのレジスタを指定することで，PCI デバイスを管理する．後者の領域は，0xCF8〜0xCFF 以外の部分であり，該当 I/O アドレスに対応付けられたデバイスの制御を行う．

全 PCI デバイスを管理する領域において，0xCF8〜0xCFB に書き込む値によって指定される部分は，PCI コンフィグレーション空間と呼ばれている．0xCFC〜0xCFF で読み書きするレジスタを指定するための各ビット内容を**図 4** に示す．バス番号は，デバイスが計算機に物理的に設置されるバスに対して付けられる番号であり，デバイス番号は，バス上のスロットに付けられる番号である．またレジスタ番号で，デバイスが持つレジスタを指定する．ファンクション番号は，複数の PCI デバイスが物理的に 1 つの PCI デバイスとして構成されるときに，バス番号，デバイス番号，レジスタ番号では区別ができないときに用いる情報である．これらの番号により，0xCFC〜0xCFF の部分で読み書きするレジスタを指定する．なお，イネーブルビットは，PCI コンフィグレーション空間に書き込む場合は 1 を指定する．

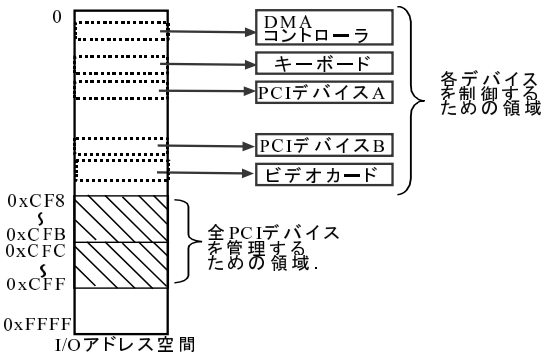

図 3 I/O アドレス空間
Fig. 3 I/O address space.

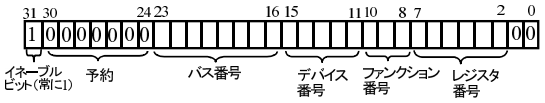

図 4 PCI コンフィグレーション空間
Fig. 4 Configuration space.

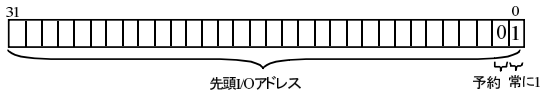

図 5 ベースレジスタの形式
Fig. 5 Base register.

4.2 PCI デバイスのマッピング

デバイスを I/O アドレス空間から制御可能とするために，各デバイスが保有するレジスタを I/O アドレス空間に対応付ける．

PCI デバイスでは，PCI コンフィグレーション空間に，対応付けたいデバイスを指定することで対応付ける．このとき，レジスタ番号はベースレジスタを意味する 0x10 を設定する．その後，0xCFC〜0xCFF の部分に，対応付ける先頭 I/O アドレスを書き込む．書き込む値の形式を**図 5** に示す．

ベースレジスタに I/O アドレスを書き込むときには，最下位ビットは I/O 空間に対応付けることを示す 1 を，下位 2 ビット目は予約ビットであることを示

す 0 を書き込む．その後 OS が I/O 命令を実行した場合，デバイスは保持するレジスタの数と先頭 I/O アドレスより，命令に応答するか否かを決定する．

また，OS もデバイスを制御するために，各デバイスの I/O アドレス幅を知る必要があるが，次の方法で把握できる．ベースレジスタは，すべて 1 のビットを書き込まれたとき I/O アドレス幅を表すために必要なビット数だけ，0 を最下位ビットから書き込む．または I/O アドレス幅を表すために必要なビット数だけ，値が 0 の読み込み専用ビットが設置されている．ただし，最下位ビットはつねに 1 である．すなわち，最下位ビットが 1，2 ビット目から 8 ビット目が 0 のとき，I/O アドレス幅は 256 である．

いずれの場合にも，ベースレジスタにすべて 1 の値を書き込み，その後読み込むことで，下位ビットの 0 となるビットを確認できる．そして，0 であるビット数に最下位ビット分の 1 を加えた数を 2 の累乗すると I/O アドレス幅を取得できる．

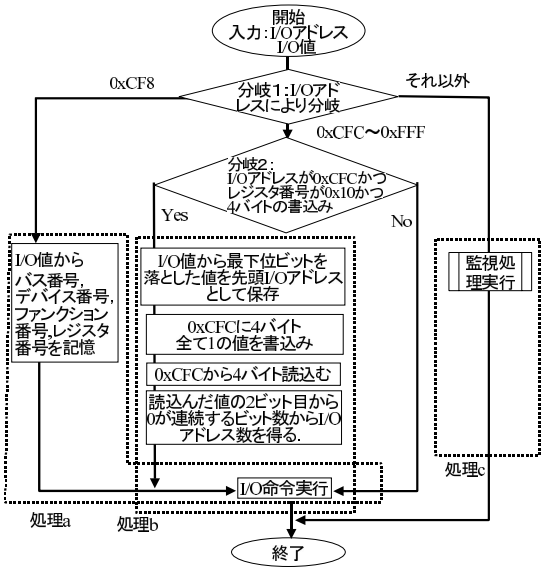

図 6 I/O アドレスを把握する処理
Fig. 6 I/O address resolution.

5. 監視プログラム構造

3 章で述べた監視構造の各部における課題について，それを解決する構造を説明しながら，対処として以下に述べる．

5.1 対　　処

（対処 1） OS が実行する I/O 命令の横取り方法

OS が I/O 命令を実行するときに，I/O 命令の把握が可能であれば，監視対象のデバイスに対する I/O 命令を一律に把握できる．このため，OS の I/O 命令を書き換え，I/O 命令実行時に，監視を行う処理を呼び出すものとする．具体的には，カーネルのソースコードを変更可能な Linux について，コード中の I/O 命令をマクロで置き換え，I/O 命令の代わりに int 命令を発行するように変更した．

（対処 2） デバイスが発生させる割込みの監視

割込み発生直後に，割込みの内容を把握可能であれば，その後の OS の制御内容によらず，割込み内容を把握できる．そこで，OS に登録されている監視対象デバイスの割込み処理を監視を行う処理に置き換え，その処理から，デバイスの割込み処理を呼び出す．

（対処 3） PCI デバイスと I/O アドレスの対応把握

OS は，各 PCI のベースレジスタに先頭 I/O アドレスを書き込むことで各 PCI デバイスを I/O アドレス空間に対応付ける．この I/O 命令を契機として，各 PCI デバイスと I/O アドレスの対応関係を把握すればよい．このときの処理を**図 6** の処理 a と図 6 の処理 b に示す．

0xCF8 に I/O が行われるとき，はじめに図 6 の処理 a で，I/O 命令の I/O 値を保存して，次に 0xCFC〜0xCFF に対して I/O が実行されるデバイスを把握する．次に 0xCFC に対する 4 バイトの I/O が実行されるとき，処理 a で保存した I/O 値のレジスタ番号が 0x10 であれば，処理 b を実行して，4.2 節で述べた方法により，各 PCI デバイスが対応付けられる I/O アドレスを知る．

（対処 4） 監視対象 PCI デバイスの指定方法

監視の実施を決定するのはユーザであるため，ユーザから OS に初期設定を与え，監視対象を決定する．具体例として，事前に用意した設定ファイルを OS ブート時に読み込ませることが考えられる．以降この初期設定を SG 表（System Generation 表）と呼ぶ．

初期設定としては，デバイスの指定，監視の有無，監視内容が考えられる．このうち，監視の有無，監視内容は任意である．そこで，デバイスの指定方法をデバイスのバス番号，デバイス番号，ファンクション番号を用いることで，すべてのデバイスを一律に扱う．バス番号は 8 ビット幅（256）だが，バスはつねに 256 存在することはないため，バス番号ごとに SG 表を作成し，バス数により SG 表のサイズを縮小可能とする．そして，各 SG 表の中で，デバイス番号，ファンクション番号を連

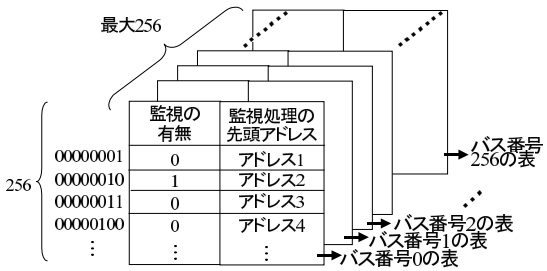

図 7 SG 表の構成
Fig. 7 System generation table.

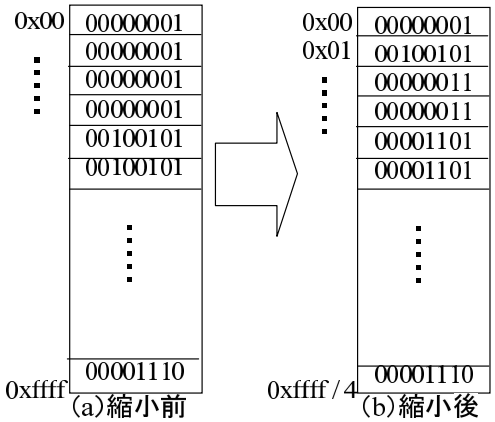

図 8 デバイス決定表の構成
Fig. 8 Device resolution table.

結させたものをインデックスとして，各デバイスを特定する．監視の有無については，ここでは監視する（1），監視しない（0）とする．

監視内容については，デバイスごとに監視有無を決定できるように，デバイスごとの監視処理の先頭アドレスとした．

以上をまとめて，SG 表を**図 7** に示す．SG 表は最大 256 枚作成され，表のインデックスは，デバイス番号，ファンクション番号の連結である．

（対処 5） 監視の実施有無の効率的な決定

I/O 命令を把握したとき，SG 表や I/O アドレスとデバイスの対応関係を用いて，ある I/O 命令が実行されるデバイスとそのデバイスに関して監視の有無を把握しなければならない．

このうち，監視の有無については SG 表に記載されている．そこで，I/O 命令に含まれる I/O アドレスと SG 表の各項目との対応をとる表を作成する．表の構成を**図 8** に示す．なお，以降この表をデバイス決定表と呼ぶ．

図 8 の (a) に示すように，デバイス決定表は，表のインデックスを I/O アドレスとして，表の項目

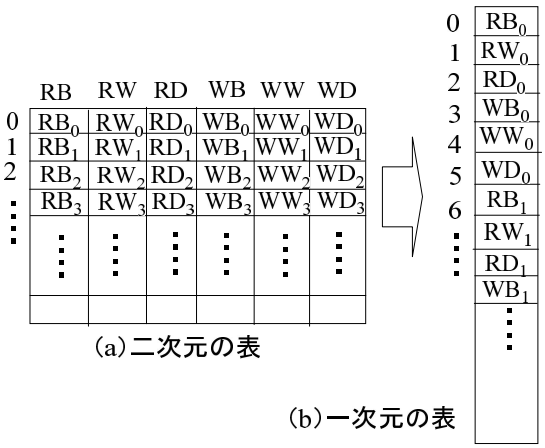

図 9 監視処理ルーチン表の構成
Fig. 9 Inspector function table.

を，SG 表のインデックスであるバス番号とファンクション番号の連結とした．このため，I/O アドレスをキーとして，デバイス決定表を検索する．そこで得られたバス番号とファンクション番号の連結をキーに SG 表を検索すると，あるデバイスの監視有無，また監視処理のアドレスが分かる．なお実際には，I/O アドレスの境界が 4 であることを利用して，図 8 の (a) を縮小した図 8 の (b) をデバイス決定表として用いる．このとき，デバイス決定表の検索方法は，I/O アドレス/4 をキーとすればよい．

（対処 6） 監視処理の粒度の細かい実行

I/O 命令が実行される PCI デバイスのレジスタごとに異なる監視処理が可能であれば，細かい監視が実施可能である．そこで，デバイスのレジスタごとに監視処理を規定する．I/O 命令には，1 バイト，2 バイト，4 バイトの読み込み/書き込みの 6 種類がある．I/O を行うバイト数により処理が異なるデバイスもあるため，I/O アドレスごとに 6 種類の処理を規定する．そして，規定した SG 表に監視処理の先頭アドレスを登録する．以降，このデバイスごとの監視処理の集合を監視処理ルーチン表と呼ぶ．**図 9** に，監視処理ルーチン表を示す．表の項目は，1 文字目が読み込み（R），書き込み（W）を表し，2 文字目はバイト数（1 バイト：B，2 バイト：W，4 バイト：D）を表す．すなわち，RB は 1 バイトの読み込みである．RBn は，デバイスの先頭 I/O アドレスから n 離れた I/O アドレスに対応するレジスタ用の処理へのアドレスである．

図 9 の (a) は二次元の表を用いて，監視処理を構

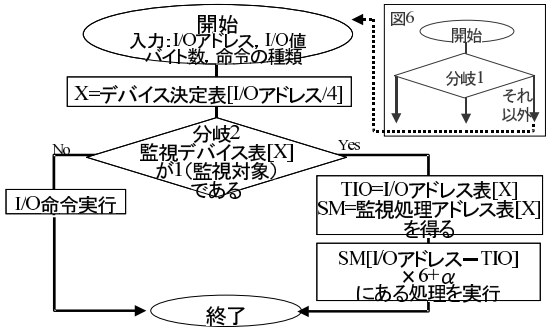

図 10 監視処理実行までの流れ
Fig. 10 Control flow of inspection.

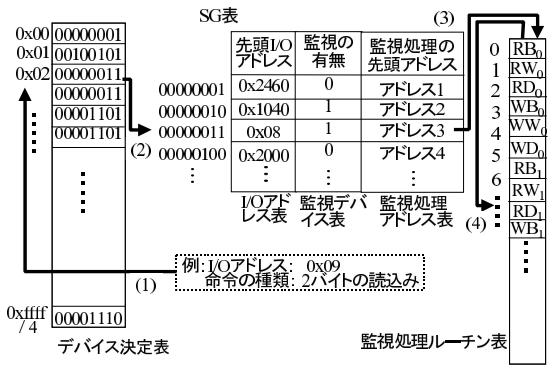

図 11 各表の関係図
Fig. 11 Relation among tables.

成するものである．表の各列に，I/O 命令の種類ごとに監視処理をまとめ，表の各行は，デバイスの先頭 I/O アドレスから n 離れた I/O アドレスに対応するレジスタの処理アドレスを格納する．この表を用いて，次のように I/O 命令から該当監視処理を得る．

まず，各 I/O 命令から命令の種類が分かるため，参照すべき列が分かる．また，各 I/O 命令の I/O アドレスと各デバイスの先頭 I/O アドレスの差分が分かれば，参照すべき行を得られる．すなわち，該当処理のアドレスを得る．

また，図 9 の (b) は図 9 の (a) を一次元の表としたものである．このときも各 I/O 命令の I/O アドレスと，各デバイスの先頭 I/O アドレスの差分より，α を並び順（RB なら 1，RW なら 2···）として，差分 $\times 6+\alpha$ で表の参照箇所が得られる．なお，図 9 の (a)，(b) ともに，各デバイスの先頭 I/O アドレスを 4.2 節で述べる方法により把握し，SG 表に項目を追加して，格納しておく．

以上より，I/O 命令を把握した後，I/O 命令が実行されるデバイスを特定し，監視対象であれば該当する監視処理を実行するためには，**図 10** の処理を行えばよい．なお，これは図 6 の処理 c に該当する．

I/O 命令を把握してから，監視処理を実行するまでの流れは，デバイス管理表を参照し，参照結果をもとに監視デバイス表，I/O アドレス表，監視処理アドレス表を参照することで，監視処理を実行する．

また，これらの表の関係を**図 11** に示す．

図 11 では，0x09 に対する 2 バイトの読み込みを例としたときの各表の関係を示す．図 7 と比較し，SG 表には，監視処理ルーチン表を検索するために，監視処理アドレス表を加えている．このときの流れは，0x09 / 4 を行い，デバイス管理表の 2 番目を参照して，SG 表の参照箇所を得る．その後，監視対象 I/O アドレス表から 0x08 を得て，監視対象アドレス表からアドレス 3 を得る．その後，アドレス 3 が示す監視処理の先頭から，(0x09-0x08) × 6 + 1 を行い，7 番目の処理を実行する．

（対処 7） 本構造の挿入方法

本構造を挿入する方法として，我々は，別途提案しているデュアル OS 方式を利用することにした[11]．6 章の実装例において述べるが，詳細は文献 11) を参照されたい．

6. 実装例および評価

6.1 デュアル OS

本機構を実装し利用する一形態として，我々が別途提案しているデュアル OS での応用を述べる[11]．

デュアル OS は，Linux を対象に 1 台の計算機上で 2 つの OS を独立に走行させる技術である．図 2 の構成をデュアル OS と対比させた様子を**図 12** に示す．デュアル OS 方式では，それぞれの OS が別々の仮想空間を持っており，独立に存在している．したがって，本機構の監視対象 OS 側への挿入部分としては，OS 切替えのためのルーチンがそれにあたる．ブートから図 12 の状態に至るまでの動作の詳細は，文献 11) を参照されたい．

ここでは，図 12 に示す OS2 が監視部分で，OS1 が監視対象の OS である．

以降では，本提案機構をデュアル OS に適用し，監視側の OS2 に占有されている NIC をもう一方の OS1 から利用した例を以下に述べ，本機構によって生じるオーバヘッドについて考察する．

6.2 擬似 NIC

デュアル OS 方式では，各 OS が共有する部分をプロセッサとタイマ割込み部分のみとしている．メモリは物理メモリの領域を 2 つに分割，I/O デバイスは

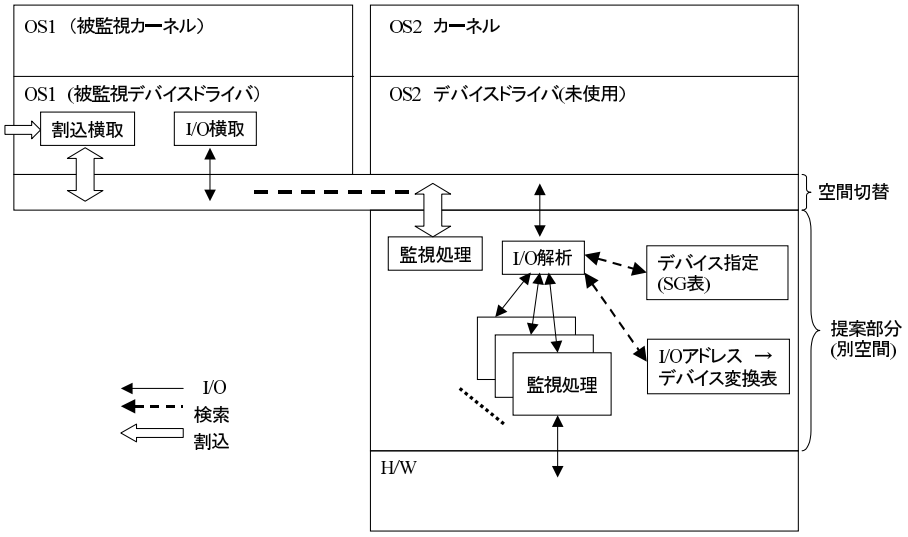

図 12 デュアル OS による PCI デバイスの監視
Fig. 12 PCI device inspection using Dual OS.

割込み発生源ごとに分割し占有させる方式をとっている．各 OS は，タイマ割込みと各 OS の占有しているデバイスからの割込みを契機に実行権を得る．

擬似 NIC は，一方の OS が占有する機器を他方の OS に提供する機能によって実現されている．これによってデュアル OS の利用法を拡大させることができる．OS はデバイスドライバを経由し I/O 命令を発行することによってハードウェアの制御を行う．このため，2 つの OS で 1 つのハードウェアを使用する場合には本機構による I/O 命令の監視が必要となる．

図 13 に擬似 NIC の構成と I/O および割込み処理の流れを示し，以降に説明する．

デュアル OS 方式において，擬似 NIC は NIC を占有する OS が持つ．OS1 の NIC ドライバが NIC に対して I/O 命令を発行しようとすると，本機構の OS1 内における実装により，OS の切替えが必要かどうか判断した後，OS 切替えが発生する．そして，擬似 NIC が I/O 命令の内容を引き継ぎ，I/O 命令の種類に応じた処理を行う．

また，処理終了等にともなう NIC からの割込みを契機に図 13 の (A)〜(D) の処理が行われる．割込み処理の内容を以下に説明する．

(A) 割込みを受ける処理，OS 切り替えの前処理，および OS を切り替える処理．
(B) ドライバで本来行われる処理と，I/O 命令発行時に OS を切り替える処理．
(C) タイマ割込みが発生するまでの OS1 での処理と OS を切り替える処理．

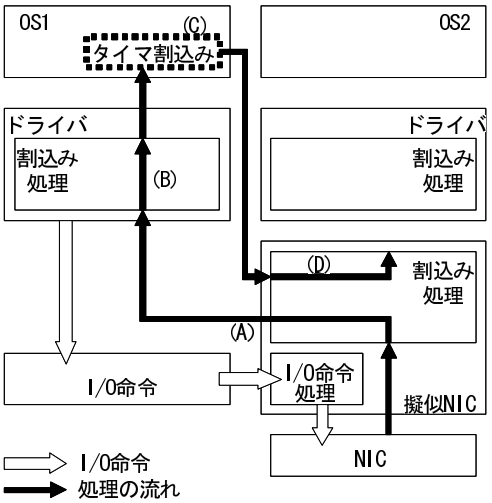

図 13 擬似 NIC の構成と割込み処理の流れ
Fig. 13 Pseudo NIC structure and interruption.

(D) OS1 からの復帰処理．

上記中で，(A) と (B) 内でも I/O 命令が発生する．また (C)，および (D) で遅延が生じるが，(C) については NIC からの割込みが入るタイミングによって遅延の大きさが異なる．タイマ割込みの場合は周期が 10 ms であるため，遅延時間は 0〜10 ms となる．以降，I/O 命令の基本的な遅延，割込み中に発生する (A)，(B) の I/O 命令での遅延，および，それらが全遅延に占める割合について評価する．

6.3 評価環境

改造を加えていない OS（以降，オリジナル）と OS1

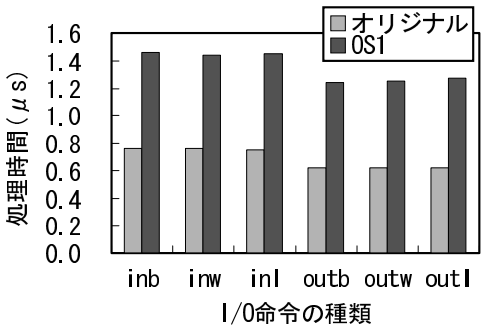

図 14 I/O 命令処理時間
Fig. 14 Delay of I/O instructions.

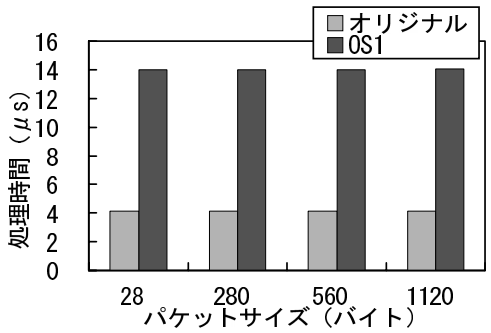

図 15 受信時間
Fig. 15 Delay of receive.

表 1 I/O 命令の遅延時間
Table 1 Delay of I/O instructions.

命令	遅延時間
inb	0.686 μs
inw	0.680 μs
inl	0.706 μs
outb	0.618 μs
outw	0.631 μs
outl	0.644 μs

の 2 つの場合について，基本 I/O 命令にかかる時間と，UDP によるパケットの受信と送信の処理時間を測定した．測定環境は，CPU: Pentium4 3.0 GHz，OS: Linux Kernel 2.4.7，NIC: 3com 3c905-TX である．測定は CPU のタイムスタンプカウント値を出力する rdtsc 命令を用いた．

6.4 基本評価

デュアル OS 方式における，基本 I/O 命令の時間について測定した．結果を**図 14** に示して説明する．

図 14 から，IN 命令，OUT 命令のどちらの場合も，アクセス単位によらず，それぞれほぼ一定の時間であることが分かる．また，本実装は，オリジナルに比べてほぼ倍の時間がかかっていることも分かる．

個々の I/O 命令における遅延時間は，0.618 μs〜0.706 μs であった．詳細を**表 1** に示す．

6.5 受信処理評価

OS1 がパケットを受信する場合の処理の流れは，図 13 に示した割込み処理の流れと同じである．また，オリジナルがパケットを受信する場合は，図 13 の (B) のドライバで行われる本来の処理のみである．

測定結果を**図 15** に示す．図 15 は，図 13 の (A)，(B)，および (D) の処理時間である．図 15 より，OS1 は，いずれのパケットサイズでも，オリジナルの場合よりも約 9.90 μs 遅くなることが分かる．その内訳を以下に示す．

(a) 割込みを受ける処理；約 0.76 μs
(b) デバイス擬似の前処理；約 1.21 μs
(c) OS を切り替える処理；約 3.99 μs
(d) I/O 時に OS を切り替える処理；約 3.19 μs
(e) OS1 からの復帰処理；約 0.75 μs

これから，(d) の I/O 時における OS 切替えの遅延が，全遅延に占める割合は，約 32%になることが分かる．ただし，この数値は，ハードウェアの種類，つまりデバイスドライバ中で発行される I/O 命令数に依存するものと考えられる．

そこで，受信割込み処理中で発行される I/O 命令数をソースコードから解析した結果，inw 命令 2 回，outw 命令 3 回であることが分かった．この回数と基本評価で得られた表 1 から，1 パケット受信にかかる総遅延時間が計算でき，$2 \times 0.680 + 3 \times 0.631 = 3.253$ となる．この値は，上記 (d) の 3.19 μs とよく一致することが分かる．つまり，割込み処理中の I/O 命令の遅延も，基本評価で得られたデータとデバイスドライバの処理内容から遅延を予測可能であるといえる．

なお，(a)，(c)，(e) は，ハードウェアの種類に依存しないため，NIC 以外の擬似ハードウェアを実装した場合も約 5.50 μs の遅延時間が I/O の頻度とは関係なく生じることも分かる．

6.6 送信処理評価

OS1 がパケットを送信する場合は，受信の場合の処理の流れに加えて，ドライバで送信処理が行われる．

測定結果を**図 16** に示す．図 16 は，ドライバでの送信処理，図 13 の (A)，(B) および (D) の処理時間である．OS1 は，いずれのパケットサイズでも，オリジナルの場合よりも約 11.98 μs 遅くなることが分かる．これは，送信の場合，デバイス擬似の前処理がないものの，受信時より I/O 命令が多く発行されているためである．内訳は，inw 命令 4 回，inl 命令 2 回，outw 命令 3 回，outl 命令 1 回である．これは，受信命令同様，基本評価のデータから計算できる遅延予測

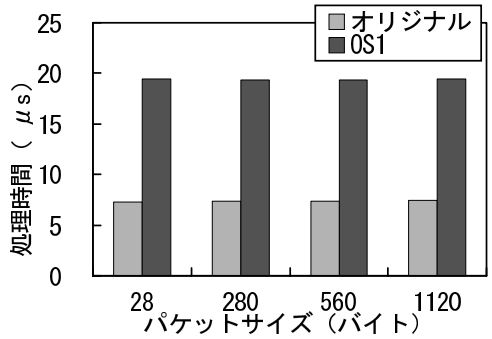

図 16 送信時間
Fig. 16 Delay of send.

とよく一致している．

6.7 考　　察

OS1 は，オリジナルに比べて 1 パケットあたり受信の場合約 9.90 μs，送信の場合約 11.98 μs 遅くなる．このうち，I/O 命令発行にともなう遅延が占める割合は，例示したケースでは，遅延全体の 45%以上になる．

100 Mbps の伝送路で通信を行った場合，1,000 バイトのパケットを送信すると，伝送時間は約 80 μs になる．この伝送時間に占める擬似 NIC による遅延時間の割合は，15%程度になる．なお，パケットサイズが大きくなるほど遅延程度は小さくなる．

また，1 つの I/O 命令の遅延は 0.6〜0.7 μs であり，ハードウェアの種類に関係なく擬似のために生じる遅延は 5.50 μs であった．これにより，提案機構を他の PCI ハードウェアに適用した場合に生じる遅延時間を予測できることが分かった．

7. おわりに

PCI デバイスを統合的に管理する機構について，実装例を含めて述べた．

本機構を用いると，監視処理が OS とデバイスの間に位置するため，他のソフトウェアへの影響を抑制し，詳細かつ効率の良い監視が可能になる．また，OS への変更は少ないため，多数の OS への対応が容易である．

本機構は，OS がデバイスを制御するための手段である I/O 命令が実行される前に特定の処理を実行するものである．また，デバイスが内部状態を通知するために発生させる割込みについて，割込み処理を目的の処理に置き換えることで，デバイスからの割込みが OS に処理される前に特定の処理を実行するものである．

応用例として，デュアル OS 方式における擬似 NIC の例を示し，I/O 操作の基本的なオーバヘッドを明らかにし，本機構を適応した場合の遅延の見積りについて考察した．

残された課題として，本機構を様々なデバイスに適用することがある．

参 考 文 献

1) Kiczales, G., Lamping, J., Maeda, C., Keppel, D. and McNamee, D.: The Need for Customizable Operating Systems, *Workshop on Workstation Operating Systems* (1993).
2) Connectix Corporation: The Technology of Virtual PC. http://connectix.com/download center/pdf/vpcw_wp/ vpcw_overviewwp_sep1301.pdf.
3) VMWare 社：VMware GSX Server. http://www.vmware.com/.
4) 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).
5) Barham, P.R., Dragovic, B., Fraser, K.A., Hand, S.M., Harris, T.L., Ho, A.C., Kotsovinos, E., Madhavapeddy, A.V.S., Neugebauer, R., Pratt, I.A. and Warfield, A.K.: Xen 2002, Technical Report UCAM-CL-TR-553, University of Cambridge (2003).
6) Sugerman, J., Venkitachalam, G. and Lim, B.H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *Proc. USENIX Annual Technical Conference* (2001).
7) Whitaker, A., Shaw, M. and Gribble, S.D.: Scale and Performance in the Denali Isolation Kernel, *Proc. 5th Symposium on Operating System Design and Implementation* (2002).
8) Basu, A., Buch, V., Vogels, W. and Eicken, T.V.: U-Net: A User-Level Network Interface for Parallel and Distributed Computing, *Proc. 15th ACM Symposium* (1995).
9) Vogels, W., Basu, A., Buch, V. and Eicken, T.V.: Incorporating Memory Management into Usesr-Level Network Interface, *A Symposium on High Performance Interconnects* (1997).
10) Birman, K.P.: Ensemble: A Tool for Building Highly Assured Networks, *DARPA Active Nets and Networking PI Meeting* (1997).
11) 田渕正樹，伊藤健一，乃村能成，谷口秀夫：二つの Linux を共存走行させる機能の設計と評価，電子情報通信学会論文誌，Vol.J88-D-I, No.2, pp.251–262 (2005).

(平成 18 年 1 月 27 日受付)
(平成 18 年 5 月 31 日採録)

乃村　能成（正会員）

平成 5 年九州大学工学部電子工学科卒業．平成 7 年同大大学院情報工学専攻修士課程修了．同年九州大学工学部助手．平成 8 年九州大学大学院システム情報科学研究科助手．平成 15 年岡山大学工学部情報工学科講師．オペレーティングシステムとソフトウェア開発環境，グループ支援環境に興味を持つ．博士（情報科学）．

山本　裕馬（学生会員）

平成 17 年岡山大学工学部情報工学科卒業．現在，同大学大学院自然科学研究科電子情報システム工学専攻修士課程在籍．オペレーティングシステムに興味を持つ．

谷口　秀夫（正会員）

昭和 53 年九州大学工学部電子工学科卒業．昭和 55 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和 62 年同所主任研究員．昭和 63 年 NTT データ通信（株）開発本部移籍．平成 4 年同本部主幹技師．平成 5 年九州大学工学部助教授．平成 8 年九州大学システム情報科学研究科助教授．平成 15 年岡山大学工学部教授．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『オペレーティングシステム』（昭晃堂）等．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．博士（工学）．

桝本　　圭（正会員）

平成 12 年明治大学理工学部情報科学科卒業．平成 14 年横浜国立大学大学院環境情報学府修士課程修了．同年（株）NTT データ入社．オペレーティングシステムの研究開発に従事．

伊藤　健一（正会員）

昭和 61 年電気通信大学電気通信学部経営工学科卒業．昭和 63 年同大学大学院修士課程修了．同年 NTT データ通信（株）入社．以来，分散処理オペレーティングシステム，マルチメディア処理装置，システムコンフィグレーション等の研究開発に従事．