特 別 研 究 報 告 書

題 目

# Mintオペレーティングシステムにおける割り込みルーティングの変更によるNICの移譲方式

指導教員

報 告 者

増田 陽介

岡山大学工学部 情報工学科

平成26年2月7日 提出

# 要約

計算機の性能の向上にともない，1 台の計算機上で複数の OS を走行させる方式が研究されている．この方式の１つに仮想計算機方式がある．しかし，仮想計算機方式で動作する OS は，計算機の仮想化によるオーバヘッドが発生し，実計算機上で動作する OS と比較して性能が低下する．また，各 OS の処理負荷の影響が存在する．そこで，1 台の計算機上で複数の Linux を走行させる方式として Mint が研究開発されている．Mint では，各 Linux がコア，メモリ，およびデバイスを分割し，直接占有制御する．これにより，Linux 間の処理負荷の相互影響をなくし，実計算機に近い性能を実現している．

Mint では動作する Linux の数は，コア数に依存している．また，Intel は，将来的に計算機に搭載するコアの数を 1000 個にすることを目標にしている．このため，Mint で同時走行する Linux の数は 10 倍以上に増加すると考えられる．一方で，1 台の計算機に搭載できるデバイスの数には限界がある．デバイスは常に使用されるわけではなく，待機時間が存在する．待機時間中にデバイスを他 Linux に移譲することで，デバイスを効率的に利用できる．

現在，Mint では Linux の既存機能である Loadable Kernel Module（LKM）を利用することで，Linux 間でデバイスを移譲できる．LKM を利用したデバイス移譲方式では，本来デバイス移譲に必要のない処理が発生し，処理時間が長大化している．また，デバイス移譲処理中は，すべての Linux が移譲対象のデバイスを使用できない．このため，デバイス移譲にかかる時間を最小限に抑えたいという要求がある．

この要求への対処として，本研究では，デバイスの 1 つである NIC を取り上げ，Mint における割り込みルーティングの変更による NIC 移譲機能を実現した．NIC 移譲には，割り込みルーティングの変更に加えて，NIC 固有の移譲処理が必要だった．

割り込みルーティングの変更による NIC 移譲機能の処理時間を計測した結果，処理時間は 1.04 ミリ秒となり，LKM を用いた移譲方式の処理時間である 8.19 ミリ秒から処理時間を約 87 %削減できた．また，割り込みルーティングの移譲機能はシステムコールで実装したため，システムコール呼出し時間により処理時間が長大化していた．システムコール呼出し時間は，処理時間全体である 1.04 ミリ秒の 96 %を占めていた．システムコール呼出し時間を除いた場合，NIC 移譲に必要な時間は約 2.0 マイクロ秒となった．

# 目次

# 図 目 次

# 表 目 次

# 第 1 章

# はじめに

計算機の性能の向上にともない，1 台の計算機状で複数の OS を走行させる方式が研究されている．この方式の 1 つに VMWare[1] や Xen[2] といった仮想計算機方式がある．しかし，仮想計算機方式で動作する OS は，計算機の仮想化によるオーバヘッドが発生する．このため，実計算機上で動作する OS と比較して性能が低下する．また，各 OS の処理負荷の影響が存在する．

そこで，計算機上で複数の Linux を独立に走行させる方式として Multiple Independent OS with New Technorogy（以降，Mint と呼ぶ）が研究開発されている．Mint では，各 Linux が CPU，メモリ，およびデバイスを分割し，直接占有制御する．これにより，Linux 間の処理不可の相互影響をなくし，実計算機に近い性能を実現した．

Mint では動作する Linux の数は，コア数に依存している．また，Intel は，将来的に計算機に搭載するコアの数を 1000 個にすることを目標としている [3]．これが実現すれば，Mint で同時走行する Linux の数は 10 倍以上に増加すると考えられる．一方で，1 台の計算機に搭載できるデバイスの数には限界がある．デバイスはつねに利用されるわけではなく，待機時間が存在する．待機時間中にデバイスを他の Linux に移譲することで，待機時間を削減し，デバイスを効率的に利用できる．

現在，Mint では Linux の既存機能である Loadable Kernel Module（以降，LKM）を利用することで，Linux 間でデバイスを移譲できる．LKM を利用したデバイス移譲では，移譲時にデバイスドライバのロードにともなうデバイスのレジスタの初期化を利用している．しかし，デバイスドライバのロード時にはレジスタの初期化の他に，デバイスドライバの登録処理というデバイス移譲に直接関係のない処理が発生する．デバイス移譲処理中は，すべての Linux がデバイスを使用できないため，デバイス移譲にかかる時間を最小限に抑えたいという要求がある．

この要求への対処として，本研究では，デバイスの 1 つである Network Intferface Card（以降，NIC と呼ぶ）を取り上げ，割り込みルーティングの変更による NIC 移譲方式を実現する．まず，Linux 間でデバイスを移譲する目的と課題について述べる．次に，割り込みルーティングの変更による NIC 移譲機能の実装方法について述べる．最後に，割り込みルーティングの変更による NIC 移譲にかかる時間を計測し，LKM を利用した NIC 移譲にかかる時間と比較した結果について述べる．

# 第 2 章

# Mint オペレーティングシステム

## 2.1 設計方針

1 台の計算機上で複数の Linux を独立走行させる方式として，Mint が研究開発されている．Mint の設計方針を以下に示す．

(1) 独立走行
すべての Linux が相互に他 Linux の処理負荷の影響を受けない方式の実現．

(2) 入出力性能の低下を最小化
すべての Linux が入出力機器の性能を十分に利用可能．

## 2.2 基本構成

Mint の構成例を図 2.1 に示し，以下で説明する．

Mint では，最初に 1 つの OS を起動し，その後別の OS を起動する．最初に起動する OS を OS ノード 0 とし，後から起動する N 番目の OS を OS ノード N と呼ぶ．

(1) CPU
コア単位で分割し，各 OS ノードは 1 つ以上のコアを占有する．

(2) 実メモリ
空間分割し，各 OS ノードは分割領域を占有する．

(3) 入出力機器
デバイス単位で分割し，各 OS ノードが仮想化によらず直接占有する．

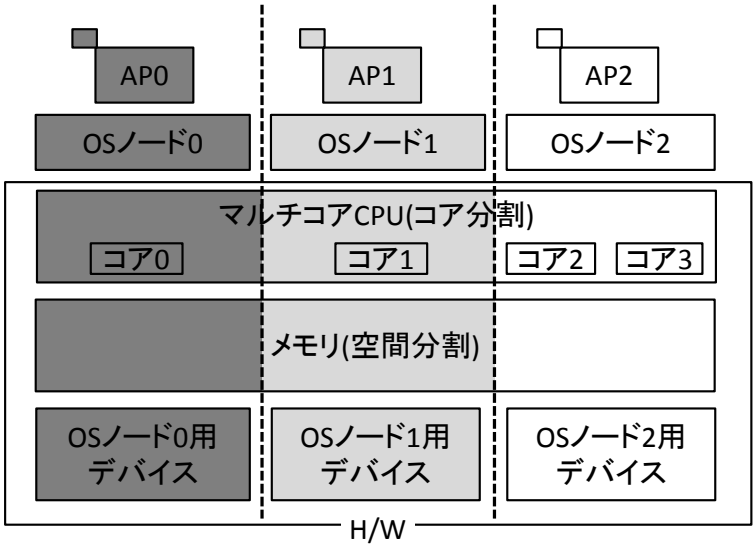


図 2.1 Mint の構成例

## 2.3 Mint における PCI デバイス分割

通常，OS ノードは起動時に PCI バスに接続されているすべての PCI デバイスを占有しようとする．PCI デバイスの占有状態は，デバイス内のレジスタの初期化により変更される．デバイス内のレジスタの初期化は，デバイスドライバをデバイスに登録する処理の中で行われる．デバイスドライバを登録する様子を図 2.2 に示し，以下で説明する．

(1) probe 処理
PCI バスに接続されているデバイスの内，デバイスドライバに対応したデバイスを探索する処理である．また，デバイスドライバに対応したデバイスであっても既にデバイスドライバを登録してあるデバイスは無視する．これは，デバイスには 1 つのデバイスドライバしか登録できないためである．

(2) attach 処理
デバイスドライバに対応したデバイスにデバイスドライバを登録し，デバイスを使用可能にする処理である．この際，デバイス内のレジスタが初期化される．これにより，デバイスの占有状態が変更される．

Mint では，probe 処理に改変を加えることで PCI デバイスを分割している．PCI デバイス A を OS ノード 0 が占有する場合を例として説明する．まず OS ノード 0 は起動時に PCI デバイス A にデバイスドライバを登録し，占有する．その後起動する他の OS ノードに対して，

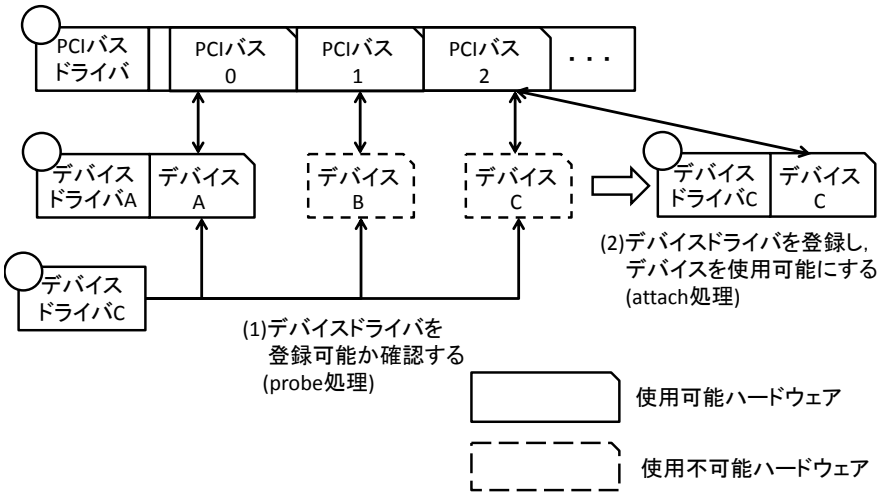


図 2.2 PCI デバイスの登録処理

PCI デバイス A の存在を隠蔽し，probe 処理の対象外とする．これにより，他の OS ノードは PCI デバイス A に対して attach 処理を実行できず，PCI デバイス A の占有状態は維持される．

# 第 3 章

# Mint におけるデバイス移譲方式

## 3.1 デバイス移譲の目的

Mint におけるデバイス移譲とは，複数の OS ノードが走行中にある状態で，ある OS ノードが占有するデバイスの占有状態を変更し，別の OS ノードが使用できるようにすることである．これによって，計算機全体におけるデバイスの利用効率の向上を目的とする．

Mint において，同時走行可能な OS ノードの数は，コア数の増加とともに増加すると考えられる．一方，1 台の計算機に搭載可能なデバイスの数には限界がある．また，デバイスは OS ノードが動作している間，つねに使用されるわけではなく，待機時間が存在する．このため，デバイスを使用していない OS ノードからデバイスを必要とする OS ノードへデバイスを移譲することで，デバイスの待機時間を削減し，デバイスの利用効率を向上できる．

## 3.2 LKM によるデバイス移譲

### 3.2.1 概要

Mint は，Linux カーネルを改変して開発されており，Linux の機能を利用できる．Linux の既存機能である LKM を利用することで，デバイス移譲を実現できる．LKM は，Linux カーネルに対する拡張機能を実現したオブジェクトで，カーネルの起動後に動的に追加削除可能である．通常，Linux カーネルの機能を追加，削除する際は Linux カーネルのソースコードを変更してビルドした後にカーネルを再起動する必要がある．LKM をロード，アンロードすることで，Linux カーネルの機能を再起動することなしに変更できる．デバイスドライバを

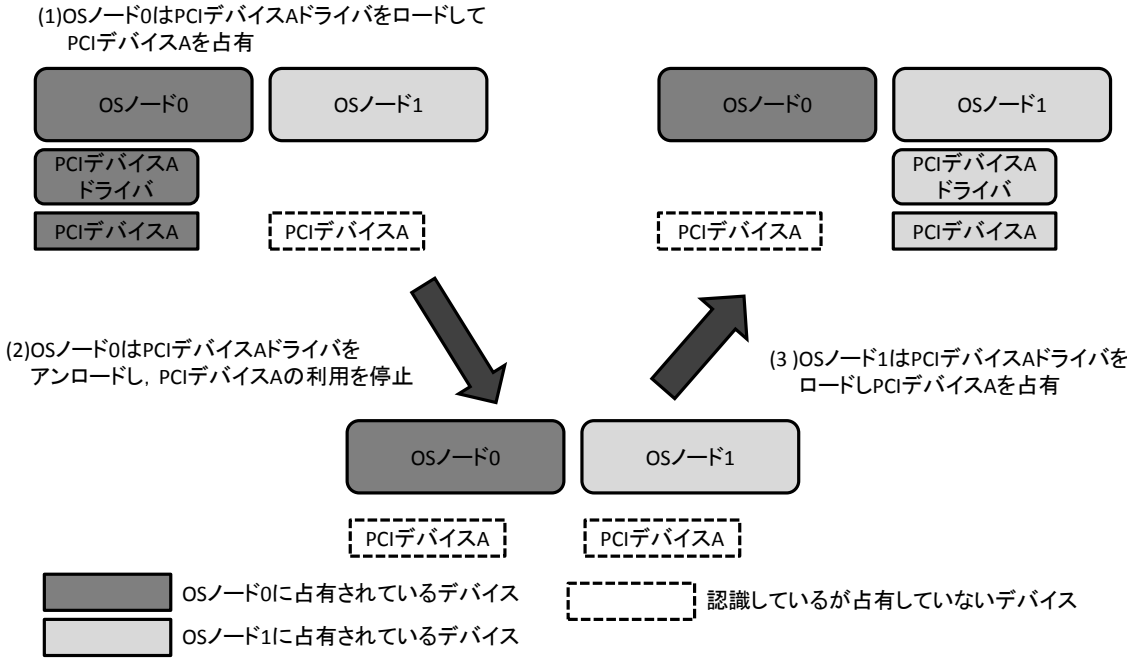


図 3.1 LKM を利用したデバイス移譲

LKM を用いてロードする際，2.3 節で述べたデバイスドライバ登録処理が発生する．このとき，デバイスドライバに対応するデバイス内のレジスタが初期化される．Mint では，LKM のロード，アンロードを利用することで OS ノード間でのデバイスを移譲を実現している．LKM を利用したデバイス移譲は，デバイス内のレジスタの初期化によって，デバイスの占有状態が変更されることを利用している．LKM を利用した PCI デバイス移譲方式について図 3.1 で示し，以下で説明する．

(1) OS ノード 0 は PCI デバイス A ドライバをロードして PCI デバイス A を占有
OS ノード 0 は PCI デバイス A ドライバをロードする．このとき．OS ノード 0 は PCI デバイス A に対して attach 処理を実行する．これにより，PCI デバイス A 内のレジスタが初期化され，PCI デバイス A が OS ノード 0 に占有される．その後，PCI デバイス A ドライバの登録が完了し，PCI デバイス A を利用可能になる．

(2) OS ノード 0 が PCI デバイス A ドライバをアンロード
PCI デバイス A ドライバをアンロードすることにより，PCI デバイス A ドライバが OS ノード 0 から削除される．これにより，OS ノード 0 は PCI デバイス A を利用不可能になる．

(3) OS ノード 1 は PCI デバイス A ドライバをロードして PCI デバイス A を占有

OS ノード 1 は PCI デバイス A ドライバをロードする．(1) と同様に，PCI デバイス A 内のレジスタが初期化される．これにより，PCI デバイス A が OS ノード 1 に占有される．その後，PCI デバイス A ドライバの登録が完了し，OS ノード 1 は PCI デバイス A を利用可能になる．

## 3.3 デバイス移譲における問題点と要求

LKM を利用したデバイス移譲は，デバイスドライバのロードにともないデバイス内のレジスタが初期化されることを利用している．デバイスドライバのロード時には，デバイス内のレジスタの初期化に加えて，デバイスドライバをデバイスに登録する処理が発生する．また，デバイスによっては，移譲後のデバイスを使用可能とするために設定を行う場合がある．例として，NIC を実際に通信可能とする際には，Physical layer や Media Independent Interface に関係するレジスタや割り込みハンドラの設定と，設定結果が NIC に正しく反映されるまで待機する処理が発生する．このため，LKM を利用したデバイス移譲方式では，デバイス移譲に必要のない処理により処理時間が長大化するという問題がある．

デバイス移譲処理中は，すべての OS ノードが移譲対象のデバイスを使用できない．このため，1 回のデバイス移譲にかかる時間が大きいと，デバイス移譲回数が増大した際，デバイスを利用できない時間が増加する．よって，デバイス移譲にかかる時間を最小限に抑えたいという要求がある．

# 第 4 章

# 高速なNIC移譲機能を実現するための課題と対処

## 4.1　割り込みルーティング

### 4.1.1　割り込み線を用いた割り込み通知法

PCI デバイス A が割り込み線を用いて，コア 0 に割り込みを通知するまでの流れを図 4.1 に示し以下で説明する．

(1) 割り込み線を通して IOAPIC のピン番号 m に割り込みを通知
PCI デバイス A は，割り込み線を通して IOAPIC のピン番号 m に割り込みを通知する．割り込み線とは，IOAPIC と PCI デバイスを直接結ぶ信号線である．

(2) ピン番号から，割り込み通知先コアとベクタ番号を算出
IOAPIC はリダイレクションテーブルの m 番目のエントリを参照し，割り込み通知先コア 0 とベクタ番号 n を求める．リダイレクションテーブルとは，ピン番号と割り込み通知先，ベクタ番号を対応付ける表である．また，ベクタ番号とは Local APIC が割り込み元を特定する際に使用する値である．

(3) 指定されたコアの Local APIC へベクタ番号 n を通知
IOAPIC は，リダイレクションテーブルから算出したコア 0 の Local APIC へ，ベクタ番号 n を通知する．これにより，PCI デバイス A が発行した割り込みはコア 0 へ通知される．

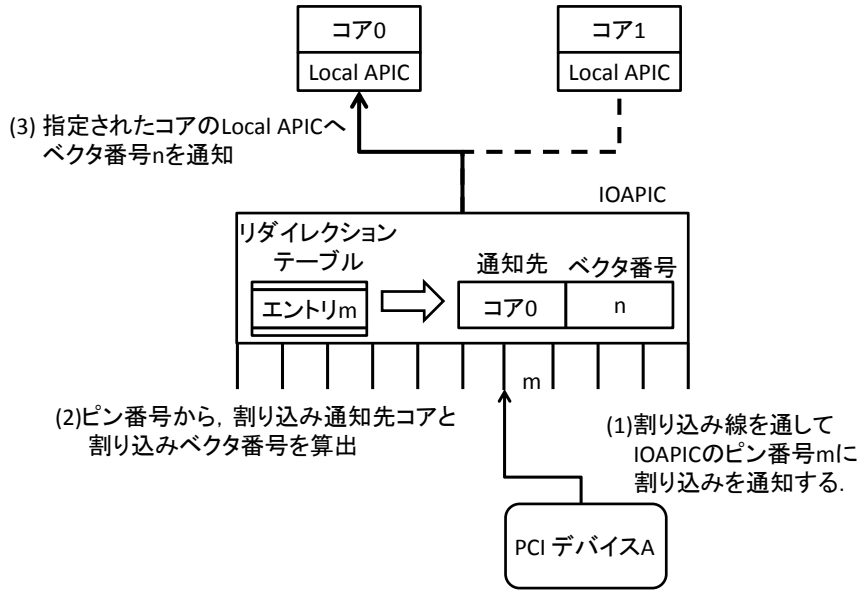


図 4.1 割り込み線を用いた割り込みのルーティング

### 4.1.2 MSI

Message Signaled Interrupt（以降，MSI と呼ぶ）は，PCI デバイスで用いられる割り込み手法である．MSI を用いた割り込みでは，デバイスは特定のアドレスに特定のデータを書き込むことで割り込みを発行する．

### 4.1.3 MSI を用いた割り込みに使用するレジスタ

PCI デバイスは，MSI を用いた割り込み発行に関わる 2 種類のレジスタを持つ．この 2 つのレジスタ，Message Address Register と Message Data Register について以下で説明する．

(1) Message Address Register
Message Address Register は，割り込み通知時にデータを書き込むアドレスを格納するレジスタである．このアドレスへの書き込みは，チップセットによって割り込みに変換され，CPU へ通知される．このレジスタの 12〜19 ビットは，割り込み通知先コアを指定する情報を表す．

(2) Message Data Register
Message Data Register は，割り込み通知時に書き込むデータの内容を格納するレジスタである．このレジスタの 0〜7 ビットは，割り込みベクタ番号を表す．

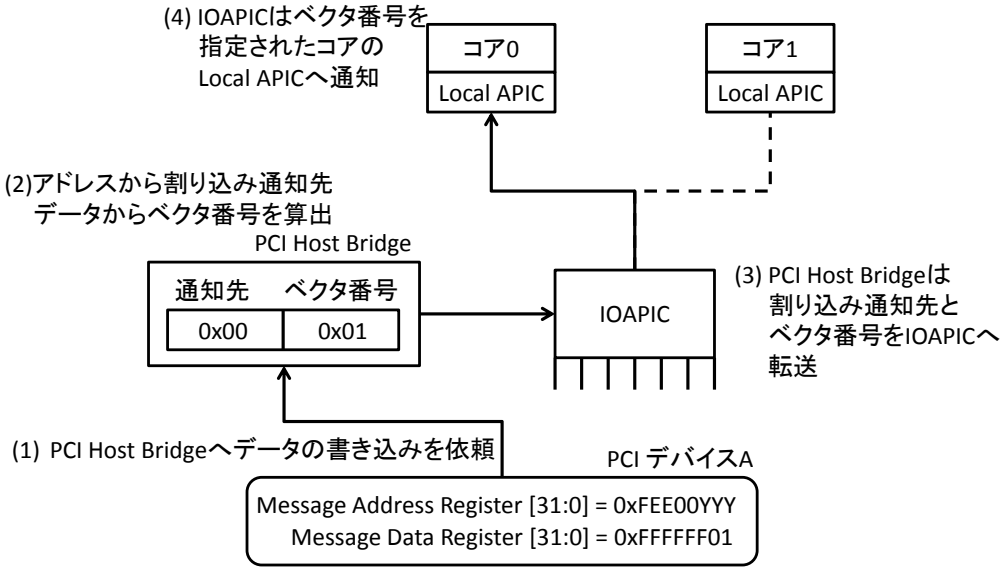


図 4.2 MSI を用いた割り込みのルーティング

MSI に関するレジスタの内容は，ユーザがカーネルに依頼することで書き換えることができる．このため，MSI では PCI デバイスの割り込み通知先，割り込みベクタ番号を，ユーザが任意に変更することができる．

### 4.1.4 MSI を用いた割り込み処理の流れ

PCI デバイス A が MSI を用いて，コア 0 に割り込みを通知するまでの流れを図 4.2 に示し以下で説明する．

(1) PCI Host Bridge へデータの書き込みを依頼
PCI デバイス A は，PCI Host Bridge に，Message Address Register に格納されているアドレスへ Message Data Register に格納されているデータを書き込むよう依頼する．PCI Host Bridge は，PCI デバイスを管理するコントローラである．

(2) アドレスから割り込み通知先，データからベクタ番号を算出
PCI デバイスから書き込み依頼を受けた PCI Host Bridge は，データの書き込み先アドレスが MSI に登録されたものだった場合，依頼された書き込み処理を，割り込みに変換する．具体的には．アドレスから割り込み通知先コア 0，データからベクタ番号 1 を算出する．

(3) PCI Host Bridge は割り込み通知先とベクタ番号を IOAPIC へ転送

(4) IOAPIC はベクタ番号を指定されたコアの Local APIC へ通知

### 4.1.5　割り込みルーティングの変更

割り込み線を用いた割り込みと，MSI を用いた割り込みは，割り込みルーティングの変更方法が異なる．それぞれの割り込みルーティングの変更方法を以下に示す．

(1) 割り込み線を用いた割り込み
IOAPIC が持つリダイレクションテーブルの内容を書き換えることで，割り込みルーティングを変更する．リダイレクションテーブルの書き換えに失敗すると，他の割り込みに関する情報が破壊されてしまう可能性がある．

(2) MSI を用いた割り込み
Message Address Register と Message Data Register の内容を書き換えることで，割り込みルーティングを変更する．デバイスのレジスタを書き換えるため，失敗してもデバイスにしか影響が出ない．

## 4.2　NIC における MSI 割り込み処理概要

### 4.2.1　NIC の機能

NIC の機能は以下の 2 つに分けることができる．

(1) パケット送信機能

(2) パケット受信機能

これらの 2 つの機能が処理対象とする OS ノードを変更することで，NIC を移譲できる．次項以降で，パケット送信処理と，パケット受信処理における NIC ドライバと NIC の処理流れを示す．本論文で移譲対象とする NIC と NIC ドライバを表 4.1 に示す．

表 4.1 移譲対象とする NIC と NIC ドライバ

| 項目名 | 対象名 |
|---|---|
| NIC | RTL8111/8168B PCI Express Gigabit Ethernet controller |
| NIC ドライバ | RTL8169 |

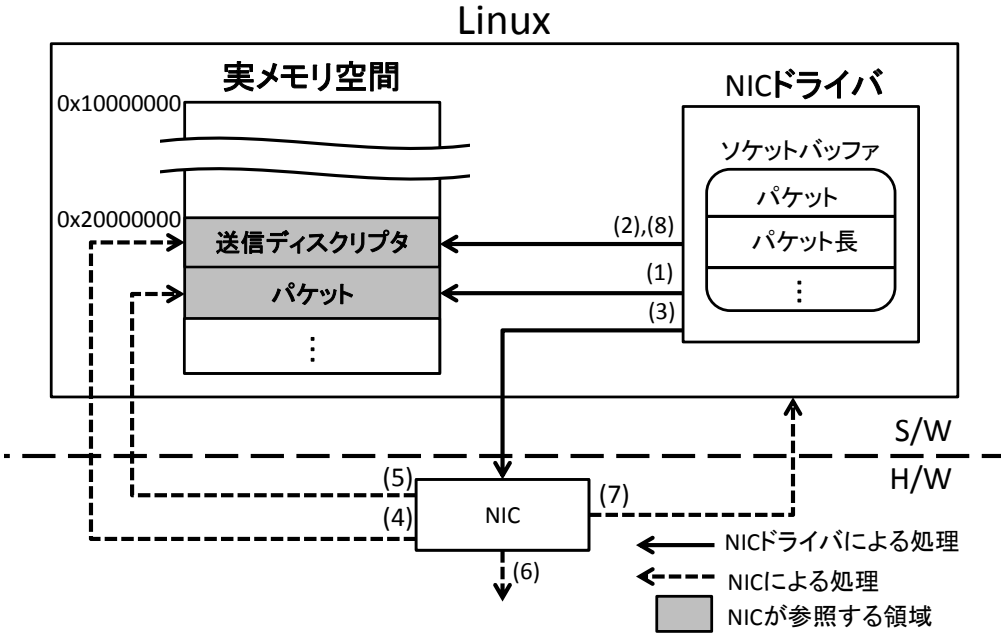


図 4.3 パケット送信機能

### 4.2.2 パケット送信機能

パケット送信機能における NIC と NIC ドライバの役割は，ソケットバッファに格納されているパケットを外部へ送信することである．ソケットバッファはパケット本体やパケット長といった情報を格納する構造体である．NIC ドライバは，プロトコル層の上位層からソケットバッファを受け取り，これを送信するよう NIC に依頼する．図 4.3 に NIC ドライバが上位層から受け取ったソケットバッファを送信する処理を示し，以下で説明する．

(1) NIC ドライバはソケットバッファに格納されているパケットを，実メモリ空間に配置する．

(2) NIC ドライバは，送信ディスクリプタに送信パケットに関する情報を格納する．送信ディスクリプタは，送信パケットに関する情報を格納するためのリングバッファである．

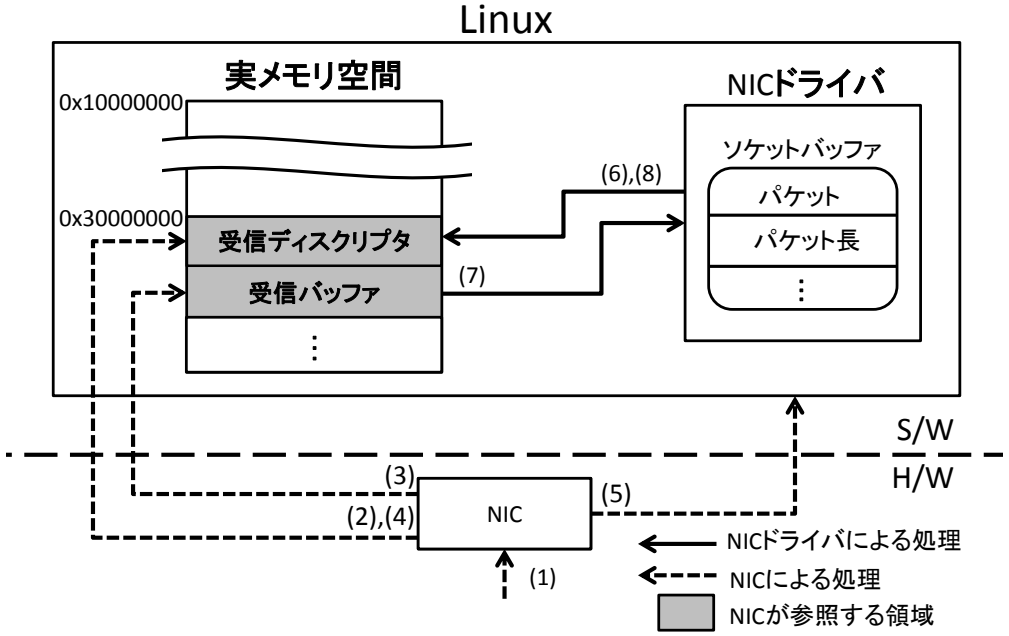


図 4.4 パケット受信機能

(3) NIC ドライバは NIC のレジスタを書き換えることで，NIC にパケット送信を依頼する．これにより NIC は送信処理を開始する．

(4) NIC は送信ディスクリプタを参照し，パケットに関する情報を取得する．この際，送信ディスクリプタの内容を更新する．

(5) NIC はパケットを取得する．

(6) NIC は送信ディスクリプタに格納されている情報をもとに，パケットを送信する．

(7) NIC は Linux へパケット送信完了割り込みを通知する．

(8) Linux は割り込みハンドラを実行し，NIC ドライバの受信処理関数を呼び出す．NIC ドライバは使用した送信ディスクリプタを初期化する．

### 4.2.3　パケット受信機能

パケット受信機能における NIC と NIC ドライバの役割は，外部から受信したパケットをソケットバッファに格納し，プロトコル層の上位層に渡すことである．図 4.4 に NIC が受信したパケットをソケットバッファに格納する処理の流れを示し，以下で説明する．

(1) NIC がパケットを受信する．

(2) NIC は受信ディスクリプタを参照し，受信バッファのアドレスを取得する．受信ディスクリプタは，受信パケットに関する情報を格納するためのリングバッファである．

(3) NIC は受信バッファにパケットを格納する．

(4) NIC は受信ディスクリプタに，受信パケットに関する情報を格納する．

(5) NIC は Linux にパケット受信完了割り込みを通知する．

(6) Linux は，割り込みハンドラを実行し，NIC ドライバの送信処理関数を呼び出す．NIC ドライバは受信ディスクリプタを参照し，受信パケットを取得する．

(7) NIC ドライバはソケットバッファを作成し，受信バッファの内容をソケットバッファに複写する．

(8) NIC ドライバは受信ディスクリプタのエントリを初期化する．

## 4.3　NIC 移譲機能を実現するための課題

(要求) の実現のための課題を以下に示す．

**(課題 1)** NIC の割り込みルーティングの変更
パケット送信完了時やパケット受信完了時に，NIC は OS ノードに割り込みを通知する．割り込み通知先を移譲先 OS ノードに変更する．

**(課題 2)** NIC が参照する送信ディスクリプタを変更
NIC と NIC ドライバは送信ディスクリプタを用いて，送信パケットに関する情報をやり取りする．送信ディスクリプタは OS ノードごとに存在する．NIC が参照する送信ディスクリプタを変更する．

**(課題 3)** NIC が参照する受信ディスクリプタを変更
NIC と NIC ドライバは受信ディスクリプタを用いて，受信パケットに関する情報をやり取りする．送信ディスクリプタと同様に，受信ディスクリプタは OS ノードごとに存在する．NIC が参照する受信ディスクリプタを変更する．

## 4.4　対処

### 4.4.1　割り込みルーティングの変更

(課題 1) への対処について述べる．4.1.3 項で示したように，MSI を用いた割り込みのルーティングは，Message Address Register と Message Data Register の内容によって決まる．したがって，Message Address Register と Message Data Register を書き換えることで，NIC が割り込みを通知する OS ノードを変更する．

### 4.4.2　NIC が参照する送信ディスクリプタの変更

(課題 2) への対処について述べる．NIC は自身が参照する送信ディスクリプタの先頭アドレスを格納するレジスタを持つ．これを書き換えることで NIC が参照する送信ディスクリプタを変更する．具体的には，以下のレジスタを書き換える．

(1) Trans Normal Priority Descriptors
NIC が参照する送信ディスクリプタの先頭アドレスを設定するレジスタ

### 4.4.3　NIC が参照する受信ディスクリプタの変更

(課題 3) への対処について述べる．NIC は自身が参照する受信ディスクリプタの先頭アドレスを格納するレジスタを持つ．これを書き換えることで NIC が参照する受信ディスクリプタを変更する．具体的には，以下のレジスタを書き換える．

(1) Receive Descriptor Start Address
NIC が参照する受信ディスクリプタの先頭アドレスを設定するレジスタ

# 第 5 章

# 実装

## 5.1 NIC 移譲機能の実装時における問題

### 5.1.1 NIC 内バッファへの参照補正

NIC と NIC ドライバはディスクリプタと呼ばれるリングバッファを用いてパケットに関する情報をやり取りする．参照するリングバッファのエントリの算出方法は，NIC と NIC ドライバで異なる．このため，NIC 移譲後に NIC と NIC ドライバが参照するエントリがずれる問題が発生する．以下に NIC と NIC ドライバのエントリ算出法を示す．

(1) NIC
NIC が参照するエントリのアドレスは，NIC 内の特定のレジスタに格納されている．このレジスタの内容は，NIC がディスクリプタを参照する度にディスクリプタの 1 エントリ分の大きさだけ加算される．これにより，NIC はディスクリプタのエントリに順次アクセスすることができる．リングバッファの終端に到達した際，このレジスタはリングバッファの先頭アドレスで初期化される．

(2) NIC ドライバ
使用したディスクリプタのエントリ数を持つ変数を利用する．この変数をリングバッファの長さで割った余りが，次に使用するエントリとなる．

NIC の参照するエントリはインクリメントで遷移する．このため，NIC が参照するリングバッファの先頭アドレスを変更しても，リングバッファの終端を参照するまで変更が反映されない．これにより，NIC と NIC ドライバが参照するエントリがずれる問題がある．

## 5.2 対処

### 5.2.1 ソフトウェアリセット

エントリがずれる問題への対処を述べる．Trans Normal Priority Descriptors と Receive Descriptor Start Address の変更は即座に反映されない．このため，NIC が参照するディスクリプタを変更するために，NIC にソフトウェアリセットをかける．ソフトウェアリセットとは，送受信処理に関わるレジスタのみを初期化する手段である．NIC の特定のレジスタのビットを立てることでソフトウェアリセットを実行し，NIC 内の送信パケットや受信パケットのカウンタを初期化する．ソフトウェアリセット後，NIC は Trans Normal Priority Descriptors と Receive Descriptor Start Address の先頭エントリから処理を再開する．まず，Trans Normal Priority Descriptors と Receive Descriptor Start Address に移譲先 OS ノードが持つディスクリプタの先頭アドレスを書き込む．その後，ソフトウェアリセットをかけることで NIC が参照するディスクリプタを変更する．

### 5.2.2 参照するエントリの同期

ソフトウェアリセット後，NIC はディスクリプタの先頭エントリを参照し，処理を開始する．NIC と NIC ドライバが参照するディスクリプタのエントリを同期させるため，NIC ドライバもソフトウェアリセット後にディスクリプタの先頭エントリから処理を開始する必要がある．NIC ドライバは変数を用いてエントリを決定する．よって，変数を初期化することで NIC ドライバが参照するエントリをディスクリプタの先頭エントリにする．具体的には，以下の変数を初期化する．

(1) cur_tx
使用した送信ディスクリプタのエントリの総数を記録する変数.

(2) dirty_tx
初期化したディスクリプタの総数を記録する変数.

(3) cur_rx
受信パケットを取り出した受信ディスクリプタのエントリの総数を記録する変数.

(4) dirty_rx
初期化したディスクリプタの総数を記録する変数.

## 5.3 NIC 移譲機能を利用するための初期化処理

OS ノード間で NIC を移譲するためには，各 OS ノードが NIC ドライバを NIC に登録し，NIC を起動する必要がある．これは，NIC の起動時に送信ディスクリプタと受信ディスクリプタが確保されるためである．NIC 移譲機能を利用する際には，NIC に対してデバイス分割を行わない．よって，最後に起動した OS ノードが NIC を占有する．OS ノード 0 と OS ノード 1 の間で，割り込みルーティングの変更による NIC 移譲機能を利用するための初期化処理について以下に示す．

(1) OS ノード 0 を起動する．

(2) OS ノード 0 は NIC ドライバをロードする．この際，OS ノード 0 は NIC のレジスタを初期化し，NIC を占有する．

(3) OS ノード 0 は NIC を起動する．

(4) OS ノード 1 を起動する．

(5) OS ノード 1 は NIC ドライバをロードする．この際，OS ノード 1 は NIC のレジスタを初期化し，NIC を占有する．これにより，OS ノード 0 は NIC を使用できなくなる．

(6) OS ノード 1 は NIC を起動する．

## 5.4 割り込みルーティングの変更による NIC 移譲

以下に Mint における割り込みルーティングの変更による NIC 移譲の手順を図 5.1 に示し，以下で説明する．

(1) OS ノード 1 は，Message Address Register（MAR）と Message Data Register（MDR）の内容を，OS ノード 1 に登録されているものに書き換える．

(2) OS ノード 1 は，Trans Normal Priority Descriptors（TNPD）と Receive Descriptor Start Address（RDSA）に OS ノード 1 が持つディスクリプタの先頭アドレスを書き込む．

(3) NIC と NIC デバイスが参照するディスクリプタのエントリを同期させる．このために，OS ノード 1 は，NIC ドライバが持つディスクリプタのエントリを決定する変数 4 つの変数 cur_tx，dirty_tx，cur_rx，および dirty_rx を初期化する．

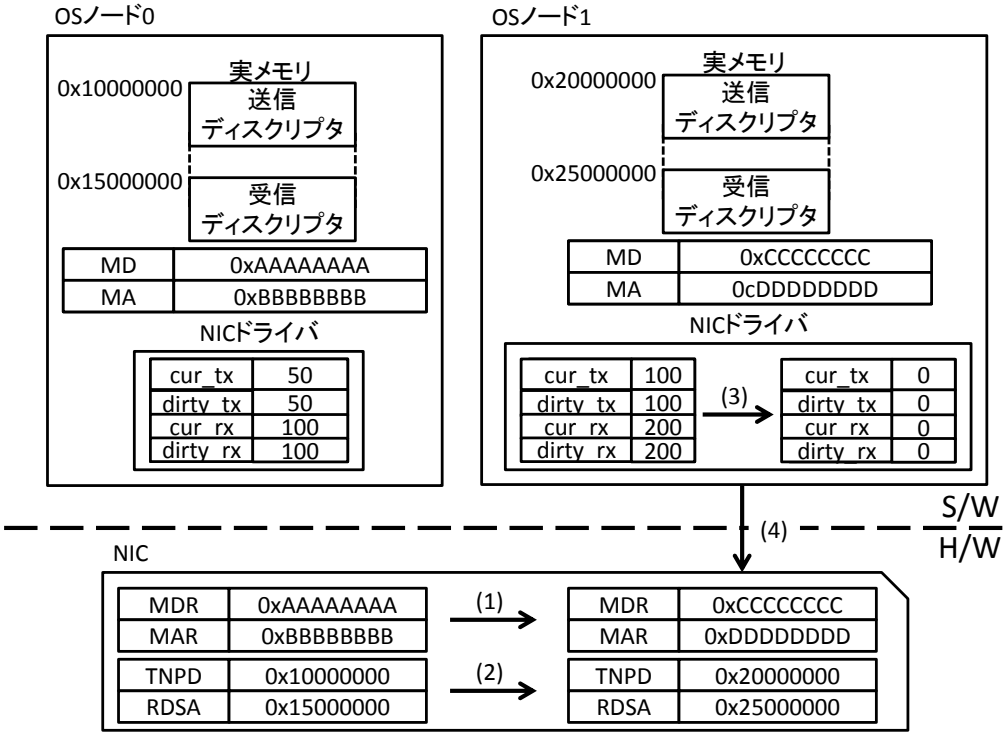


図 5.1 割り込みルーティングの変更による NIC 移譲の処理流れ

(4) OS ノード 1 は，NIC が参照する送信ディスクリプタ，受信ディスクリプタを変更するため NIC にソフトウェアリセットをかける．

# 第 6 章

# 評価

## 6.1 評価概要

本章では，割り込みルーティングの変更による NIC 移譲方式の性能を評価するため，LKM を利用した NIC 移譲方式と割り込みルーティングの変更による NIC 移譲方式の処理時間を比較評価する．評価環境を表 6.1 に示す．移譲時間の測定方法として，LKM を利用した NIC 移譲方式では，ドライバをロード，アンロードするコマンドである modprobe と rmmod の実行時間の合計を移譲時間として測定する．また，割り込みルーティングの変更による NIC 移譲方式では，システムコールに実装した NIC 移譲機能の処理が終了するまでの時間を移譲時間として測定する．なお，測定には Time Stamp Counter（TSC）レジスタを使用する．

## 6.2 LKM を利用した NIC 移譲方式

### 6.2.1 NIC ドライバのロード

モジュールをロードする処理時間の内訳を表 6.2 に示し，時間がかかっている処理について以下で説明する．

(1) システムコール呼出し処理
モジュールをロードするシステムコールである init_module を呼び出す処理である．モジュールを実メモリ上に読み込む処理が含まれるため，処理時間が長大になる．

(2) probe 処理
デバイスドライバに対応するデバイスを探索する処理である．

表 6.1 評価環境

| 項目名 | 環境 |
|---|---|
| OS | Fedora 14 x86_64(Linux Kernel 3.0.8) |
| CPU | Intel(R) Core(TM) Core i7-870 @ 2.93GHz |
| NIC | RTL8111/8168B PCI Express Gigabit Ethernet controller |
| NIC ドライバ | RTL8169 |

表 6.2 NIC ドライバのロード処理時間の内訳

| 処理手順 | 時間 (ms) |
|---|---|
| (1) システムコール呼出し処理 | 2.63 |
| (2)probe 処理 | 0.58 |
| (3)attach 処理 | 1.58 |
| 合計 | 4.79 |

(3) attatch 処理
デバイスにデバイスドライバを登録する処理である．デバイス内のレジスタを初期化し，デバイスの占有状態を変更する．また，ドライバの情報をカーネルに登録する．attach 処理の処理時間である 1.58 ミリ秒のうち，1.57 ミリ秒は NIC ドライバ内で宣言されている初期化関数の処理時間であった．初期化関数では，デバイスの初期化に加えて，メモリ領域の確保や割り込みの登録を行っていた．attach 処理の処理時間は，NIC ドライバの実装に大きく依存していると考えられる．

### 6.2.2 NIC ドライバのアンロード

モジュールをアンロードする処理時間の内訳を表 6.3 に示し，時間がかかっている処理について以下で説明する．

(1) システムコール呼出し処理
モジュールをアンロードするシステムコールである delete_module を呼び出す処理である．

表 6.3 NIC ドライバのアンロード処理時間の内訳

| 処理手順 | 時間 (ms) |
|---|---|
| (1) システムコール呼出し処理 | 1.34 |
| (2)NIC の使用の停止 | 2.04 |
| (3) ドライバの削除 | 0.02 |
| 合計 | 3.40 |

(2) NIC の使用の停止
NIC ドライバをアンロードする前に，NIC の使用を停止する必要がある．このために，NIC 内のレジスタの初期化，割り込み番号の解放といった処理を行う．NIC の初期化の際，NIC 内のレジスタを書き換えるため，多数の IO 処理を行う．また，NIC が停止可能か否かを確認する処理を行う．これにより，処理時間が長大になっている．

(3) ドライバの削除
カーネル内のデバイスに関する情報を削除する．

## 6.3 割り込みルーティングの変更による NIC 移譲方式

割り込みルーティングの変更による NIC 移譲に必要な処理時間の内訳を表 6.4 に示し以下で説明する．

(1) システムコール呼出し処理
割り込みルーティングの変更による NIC 移譲機能は，システムコールで実装している．移譲機能を実行する際，システムコールを呼び出す処理が発生する．

(2) デバイスの探索
PCI バス ID をもとに，移譲対象とするデバイスの情報を取得する．

(3) 割り込みルーティングの変更
NIC 内の MSI Message Register と MSI Data Register を書き換えて，割り込み通知先を変更する．IO 処理を 2 回行うだけなので，非常に高速な処理である．

(4) NIC 固有の移譲処理
NIC が参照する送信ディスクリプタと受信ディスクリプタを変更する．このとき，NIC

表 6.4 割り込みルーティングの変更による NIC 移譲方式の処理時間の内訳

| 処理手順 | 処理時間 (ms) |
|---|---|
| (1) システムコール呼出し処理 | 1.02 |
| (2) デバイスの探索 | 0.01 未満 |
| (3) 割り込みルーティングの変更 | 0.01 未満 |
| (4)NIC 固有の移譲処理 | 0.02 |
| 合計 | 1.04 |

表 6.5 NIC 移譲方式の処理時間の比較

| 移譲方式 | 処理時間 (ms) |
|---|---|
| (1)LKM を利用したデバイス移譲方式 | 8.19 |
| (2) 割り込みルーティングの変更による移譲方式 | 1.04 |
| (1) - (2) | 7.15 |

にソフトウェアリセットをかける必要がある．ソフトウェアリセットをかけた後，NIC が使用可能になった事を確認するまでの待ち時間が含まれている．

## 6.4 NIC 移譲時間の比較

LKM を利用した NIC 移譲方式と割り込みルーティングの変更による NIC 移譲方式の処理時間を計測し，比較した結果を表 6.5 に示す．

LKM を利用したによる移譲方式の処理時間が 8.19 ミリ秒だったのに対して，割り込みルーティングの変更による移譲方式の処理時間は 1.04 ミリ秒となり，約 87 %処理時間を削減できた．LKM を利用した移譲方式では，NIC のレジスタを初期化する処理に時間がかかっていた．これは，NIC を初期化する際に，多数の IO 処理を行っているためだと考えられる．また，NIC ドライバのロードとアンロード処理を行うシステムコール呼出しに時間がかかっていた．特に，NIC ドライバのロード時は NIC ドライバを読み込む処理が必要なため，処理時間が長大化していた．これらの処理を除き，初期化するレジスタを最小限に抑えることで，高速な NIC 移譲を実現できた．

今回，割り込みルーティングの変更による NIC 移譲機能をシステムコールで実装した．このため，NIC 移譲機能にはシステムコールの呼出し時間が含まれている．システムコールの呼出し時間は，移譲処理時間である 1.04 ミリ秒の内の 1.02 ミリ秒であり，全体の約 96 %を占めている．システムコールの呼出し時間を除き，移譲機能のみの処理時間を計測した場合，移譲時間は約 2.0 マイクロ秒となった．このため，システムコールによらない実装をすることで，さらに高速な NIC 移譲を実現できる．

# 第 7 章

# おわりに

本論文では，Mint における割り込みルーティングの変更による NIC の移譲方式について述べた．まず，Mint における OS ノード間でのデバイス移譲の目的について述べた．次に，割り込みルーティングの変更による NIC 移譲方式を示した．最後に，割り込みルーティングの変更による NIC 移譲機能を実装する方法について示した．その後，実装した割り込みルーティングによる NIC 移譲機能の処理時間を計測し，LKM を用いた移譲の処理時間と比較した．

割り込みルーティングの変更による NIC 移譲方式では，割り込みルーティングの変更に加えて，NIC が参照する送信ディスクリプタ，受信ディスクリプタを変更するという NIC 固有の移譲処理が必要だった．LKM を利用した NIC 移譲の処理時間は 8.19 ミリ秒だった．一方，割り込みルーティングの変更による NIC 移譲の処理時間は，1.04 ミリ秒であり，LKM を利用した場合と比較して処理時間を約 87 %削減できた．LKM を利用した移譲方式で時間がかかっていた処理は，システムコールの呼出し処理と NIC の初期化処理だった．NIC の初期化処理は多数の IO 処理を行っているため，処理時間が長大化していると考えられる．

また，割り込みルーティングの変更による移譲機能の処理時間 1.04 ミリ秒のうち 1.02 ミリ秒は，システムコール呼出しの時間だった．これは，システムコールとして NIC 移譲機能を実装したためである．システムコール呼出し時間を除いた場合，NIC 移譲に必要な時間は約 2.0 マイクロ秒となった．このため，システムコールによらない実装をすることでより高速な NIC 移譲を実現できる．

# 謝辞

本研究を進めるにあたり，懇切丁寧なご指導をしていただきました乃村能成准教授に心より感謝の意を表します．また，研究活動において，数々のご指導やご助言を与えていただいた谷口秀夫教授，山内利宏准教授ならびに後藤佑介助教に心から感謝申し上げます．

また，日頃の研究活動において，お世話になりました研究室の皆様に感謝いたします．

最後に，本研究を行うにあたり，経済的，精神的な支えとなった家族に感謝いたします．

# 参考文献

[1] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, Proc. of the GeneralTrack: 2002 USENIX Annual Technical Conference, pp.1-14, (2001).

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and AndrewWarfield: Xen and the Art of Virtualization, Proc. of the 19th ACM Symposium on Operating Systems Principles, pp.164-177, (2003).

[3] Jack Clark: Experimental Intel chip could scale to 1,000 cores (online), available from 〈http://www.zdnet.com/experimental−intel−chip−could−scale−to−1000−cores−4010021129/〉 (accessed 2014-01-30).