修 士 論 文

題 目

Mintオペレーティングシステムにおける NICのコア間移譲方式

指導教員

報 告 者

増田 陽介

岡山大学大学院 自然科学研究科 電子情報システム工学専攻

平成28年2月3日 提出

要約

計算機資源を効率的に利用するため，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．仮想計算機方式では，OS 間で 1 つの I/O デバイスを共有可能だが，ハイパーバイザによる処理負荷により I/O 性能が低下する．一方，Mint オペレーティングシステムは，仮想化によらず I/O デバイスを分割占有することで，性能低下を抑制できる．しかし，OS と I/O デバイスの関係は，起動時に静的に固定されるため，I/O デバイスの接続形態の変更のしやすさは仮想計算機方式に及ばない．そこで本稿では，移譲方式，つまり Mint において I/O デバイスを占有する OS を動作中に切替える方式を用いた I/O デバイスの時分割を提案する．具体的には，I/O デバイスの 1 つである NIC を対象に，Mint において NIC を占有する OS を動作中に短時間で切り替えることで，複数 OS で NIC を利用する．Mint における NIC の移譲方式として，割込ルーティングの変更による移譲方式を提案し，Loadable Kernel Module(LKM) を利用した移譲方式と比較する．NIC を時分割することを想定した場合，1 回の移譲にかかる時間を最小限にしたいという要求がある．LKM を利用した移譲方式は，NIC の制御に必要な情報をすべて初期化する必要があるため，秒単位で通信が停止する．一方，割込ルーティングの変更による移譲方式は，初期化する情報を最小限にすることでマイクロ秒単位での高速な移譲を実現した．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機に搭載される CPU のコア数や実メモリ量が増加し，計算機の性能が向上している．これらの計算機資源を効率的に利用するため，1 台の計算機上で複数のオペレーティングシステム (以降，OS) を同時走行させる方式の研究が活発に行われている．代表的なものとして，仮想計算機方式 (以降，VM 方式) があり，代表的な研究として，Xen[1] や VMware[2] がある．しかし，VM 方式では仮想化により実計算機に比べて性能が低下する [3][4].

I/O デバイス使用時のオーバヘッドを削減する仮想化支援方式として，PCI パススルー [5] と SR-IOV がある．これらは，ハードウェアレベルの仮想化支援機能であり，ゲスト OS がハイパーバイザの仲介なしで I/O デバイスを使用できる．ハイパーバイザの仲介によるオーバヘッドを削減することで，I/O 性能の低下を低減する．しかし，実計算機と同等の I/O 性能を獲得するには至っていない [6][7]．また，高価な I/O デバイスを必要とする．

そこで，我々は実計算機に近い性能で複数の OS を走行可能な Mint[8] を開発している．Mint は Linux をベースに開発されており，1 台の計算機上で複数の Linux を同時走行させる方式である．Mint では，仮想化によらず各 OS を実計算機上で直接走行させることで実計算機に近い性能での OS の走行を実現している．また，OS ごとの独立性を実現しており，各 OS は互いに処理負荷の影響を与えない．

Mint で同時走行可能な OS の数は，CPU のコア数の増加に伴い増加する．一方で，計算機に搭載可能な I/O デバイスの数には限界がある．そこで，I/O デバイスを複数の OS で共有したいという要求がある．

I/O デバイスの 1 つであるネットワークインタフェースカード (以降，NIC) を例にすると，VM 方式は Open vSwitch[9] や SR-IOV を利用することで NIC を複数 OS で共有できる．しかし，仮想化によるオーバヘッドにより I/O 性能が低下する．一方，Mint は I/O デバイス

を実計算機に近い性能で利用できる．しかし，I/O デバイス単位で分割占有するため，I/O デバイスを複数 OS で共有できない．

Mint において，I/O デバイスを共有せずに複数 OS で利用する方法として，I/O デバイスを占有する OS に I/O を依頼する方法がある．この方法の場合，処理依頼数の増加に伴い I/O 性能が低下する場合がある．別の方法として，I/O を必要とするプロセスそのものを I/O デバイスを占有する OS に移動させる方法がある．Mint は OS 間の独立性が高く，プロセスのコンテキストを共有データとして持っておくことができない．このため，プロセスの移動はオーバヘッドが大きい．また，上記の 2 つの方法は，I/O デバイスを実行する OS ノードが停止した場合，同時走行するすべての OS の I/O が停止する問題がある．そこで，Mint において，OS 間で I/O デバイスを移譲し，I/O デバイスを時分割する手法を提案する．

I/O デバイスの 1 つである NIC を対象とした移譲方式として，Loadable Kernel Module (以降，LKM) を利用した移譲方式がある．LKM を利用した移譲方式は，Linux の標準機能を利用した手法であるため，幅広いデバイスに対して適用可能であるという利点があるものの，移譲処理のオーバヘッドが大きいという欠点がある．たとえば NIC の移譲においては，NIC 制御に必要な情報のうち，NIC ドライバだけでなく，ネットワークプロトコルスタックや NIC 本体が持つ情報をすべて初期化する必要があり，移譲処理に秒単位の時間がかかる．

NIC を時分割することを想定した場合，数秒から数百ミリ秒の間隔で NIC を移譲する必要がある．このため，移譲処理に秒単位の時間がかかる LKM を利用した移譲方式は，NIC の時分割には適さない．そこで，本稿では割込ルーティングの変更による移譲方式を提案する．割込ルーティングの変更による移譲方式は，移譲処理が複雑になるものの，初期化する情報量を削減することで，移譲時のオーバヘッドを削減し，マイクロ秒単位の移譲を実現する．

第 2 章

Mint とは

2.1 設計方針

Mint は Linux をベースに開発されており，仮想化によらず 1 台の計算機上で複数の OS を独立に走行させる方式である．本稿では Mint を構成する OS を OS ノードと呼ぶ．Mint の設計方針として，以下の 2 つがある．

(1) 全 OS ノードが相互に処理負荷の影響を与えない．

(2) 全 OS ノードが入出力性能を十分に利用できる．

2.2 構成

1 台の計算機上でプロセッサ，メモリ，および I/O デバイスといったハードウェア資源を効果的に分割し，それぞれの OS ノードで占有する．図 2.1 に Mint の構成例を示し，プロセッサ，メモリ，および I/O デバイスの分割と占有方法について以下で説明する．

プロセッサ は，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．

メモリ は，空間分割し，各 OS ノードに分割領域を分配する．

I/O デバイス は，I/O 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制御する．

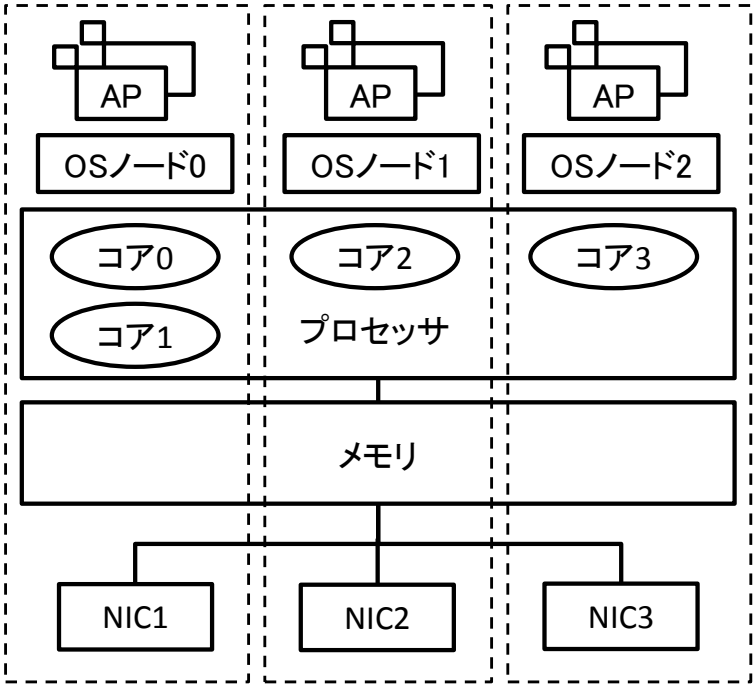

図 2.1 Mint の構成例

2.3 I/O デバイス制御のコア間移譲

Mint では，OS ノードごとの独立性が高く，I/O 性能は仮想化に比べて高い．しかし，OS ノードと I/O デバイスの関係は起動時に静的に固定されるため，I/O デバイスの接続形態の変更のしやすさは，仮想計算機方式に及ばない．そこで，I/O デバイスを共有することなく，複数の OS ノードで使用する方式として，以下の 3 つの案がある．

(案 1) I/O デバイスを占有する OS ノードへ I/O を依頼する．

(案 2) I/O デバイスを占有する OS ノードへ I/O が必要なプロセスそのものを移動する．

(案 3) I/O デバイス制御を短い間隔でコア間移動し，時分割する．

案 1 は I/O デバイスを占有しない OS ノードは，I/O デバイスを占有する OS ノードに I/O を依頼する方式である．I/O デバイスを占有する OS ノードは，他の OS ノードから依頼された I/O を実行し，結果を返却する．このため，I/O の依頼によるオーバヘッドが発生し，I/O 性能が低下すると予想される．1 台の計算機上で複数 OS を同時走行させる SIMOS で採用されている方法である [10].

案 2 は I/O を必要とするプロセスをすべて I/O デバイスを占有する OS ノードに移動し実行する方式である．この方式の問題点として，プロセスが持つデータ全てを移動する必要があり，プロセスの移動時のオーバヘッドが大きい．これは，Mint は，各 OS ノードの独立性が高くメモリ領域を完全に分割していることにより，プロセスのコンテキストを共有できないためである．また，両案ともに，I/O デバイスを占有する OS ノードが停止した場合，全ての OS ノードが I/O 処理を実行できなくなる．

案 3 は I/O デバイスの占有と解放を行い．OS ノード間で I/O デバイスの制御を移譲することで，I/O デバイスを時分割する方式である．OS ノードは I/O デバイスを占有して使用するため，実計算機に近い I/O 性能を実現できる．頻繁に移譲する必要があるため，移譲時間によっては I/O 可能な時間が低下することが考えられるものの，他の OS ノードに影響されず I/O を実行できるため，全 OS ノードが相互に処理負荷の影響を与えないという Mint の設計方針を保ったまま，複数の OS ノードで I/O デバイスを使用できる．以上から，I/O デバイス共有することなく複数 OS ノードで I/O デバイスを使用する方法として案 3 を実現する．以降で，I/O デバイスの 1 つである NIC の制御をコア間移譲する方式について述べる．

本稿では，以下の 2 つの移譲方式を提案し，比較する．

(1) LKM を利用した移譲方式

(利点) Linux の既存機能である LKM を利用した移譲方式であり，カーネルを改変する必要がない．

(欠点) 移譲時に NIC ドライバの挿抜と NIC ハードウェアの初期化を行うため，低速である．

(2) 割込ルーティングの変更による移譲方式

(利点) NIC 移譲に必要な情報のみを初期化するため，高速である．

(欠点) 初期化する情報の選別が必要である．

割込ルーティングの変更による移譲方式は，LKM を利用した移譲方式と比較して高速に移譲できる．しかしながら，NIC 移譲時に初期化の必要な情報を選別する必要がある．そこで，まず LKM を利用した移譲方式の処理流れを調査し，NIC の制御を取得するために初期化の必要な情報の全体像を把握する．また，NIC 移譲時間を測定し，低速であることを確認する．その後，初期化する情報量を削減した高速な移譲方式である割込ルーティングの変更による移譲方式について示し，移譲時間を比較する．

第 3 章

LKMを利用した移譲方式

3.1 LinuxにおけるNICの制御方法

Mint は，Linux と同等の LKM 機能を持つため，LKM のロード/アンロードによって，デバイスの着脱が可能である．これは，OS ノード間で LKM の着脱を競合することなく行えば，デバイスの移譲が可能であることを意味する．Linux における NIC の LKM ドライバ初期化には，以下の処理が発生する．

(1) NIC ドライバのロード

(2) 通信 IF の初期化

(1) により，NIC ドライバは NIC の制御に必要な情報を初期化する．(2) は，通信インタフェース (以降，通信 IF) の設定である．NIC ドライバは，通信プロトコルスタックに自らを通信 IF として登録することで，通信デバイスによる通信機能を提供する．

3.2 NICドライバのロードの有無によるNICの分割

Mint で走行する OS ノードは，3.1 節で示した処理手順を各 OS ノードが実行することで，NIC の制御を取得し，NIC を通信 IF として利用できる．NIC を使用しない OS ノードは NIC ドライバをアンロードし，NIC の制御に必要な情報を削除する．これらを競合することなく行えば，デバイスの移譲そのものは Linux の枠組の中で実現可能である．しかし，これらは，I/O デバイスのハードウェアやドライバにまつわるソフトウェアの大部分を初期化対象とするため，移譲にかかるオーバヘッドが大きいと考えられる．そこで，Linux が NIC の制

御を取得するために実行する処理手順を分析し，LKM 着脱による移譲のオーバヘッドの詳細を次に分析する．

3.3 NIC の制御に必要な情報

Linux が NIC の制御に必要な情報について図 3.1 に示す．制御においては，カーネル，NIC ドライバ，NIC ハードウェアの 3 つが協調して処理を行う．それぞれの間に相互共有すべきデータ構造やインタフェースが存在する．

カーネルから NIC ハードウェアへのインタフェースは，pci_device を介して行われる．PCI 設定レジスタのアドレスや割込の情報 (図中 msi_list) を保持するテーブルからなる．カーネルは，NIC ハードウェアを検出した時点に pci_device を初期化し，カーネルの管理リストに繋ぐ．

カーネルから NIC ドライバへのインタフェースは，pci_driver と net_device を介して行われる．pci_driver は，デバイスドライバが共通に持つべき操作のテーブルで，net_device は，ネットワークデバイスが共通に保持すべき操作やデータのテーブルである．pci_driver と net_device は NIC ドライバ初期化の際に NIC ドライバ側が確保してカーネルに返却され，カーネルの管理リストに繋がれる．

NIC ドライバから NIC ハードウェアへのインタフェースは，NIC ハードウェアのアーキテクチャによって差異があるが，NIC 固有レジスタ，送受信バッファとその領域管理用ポインタ，NIC ドライバ自身が持つ NIC ハードウェア固有の管理情報 (図中では，rtl8169_private) がある．NIC ドライバは，初期化時にカーネルから pci_device の情報が与えられ，それに基づいて NIC ハードウェアのレジスタにアクセスしつつ，rtl8169_private や送受信バッファを確保および初期化する．

3.4 NIC の制御を取得する処理手順

Linux が NIC の制御を取得するために実行する処理手順を以下に示す．

(1) NIC の検出
カーネルは NIC の PCI 設定レジスタにアクセスし，NIC ハードウェアを検出する．その後 pci_device を作成し，pci_device リストに登録する．

(2) NIC ドライバのロード

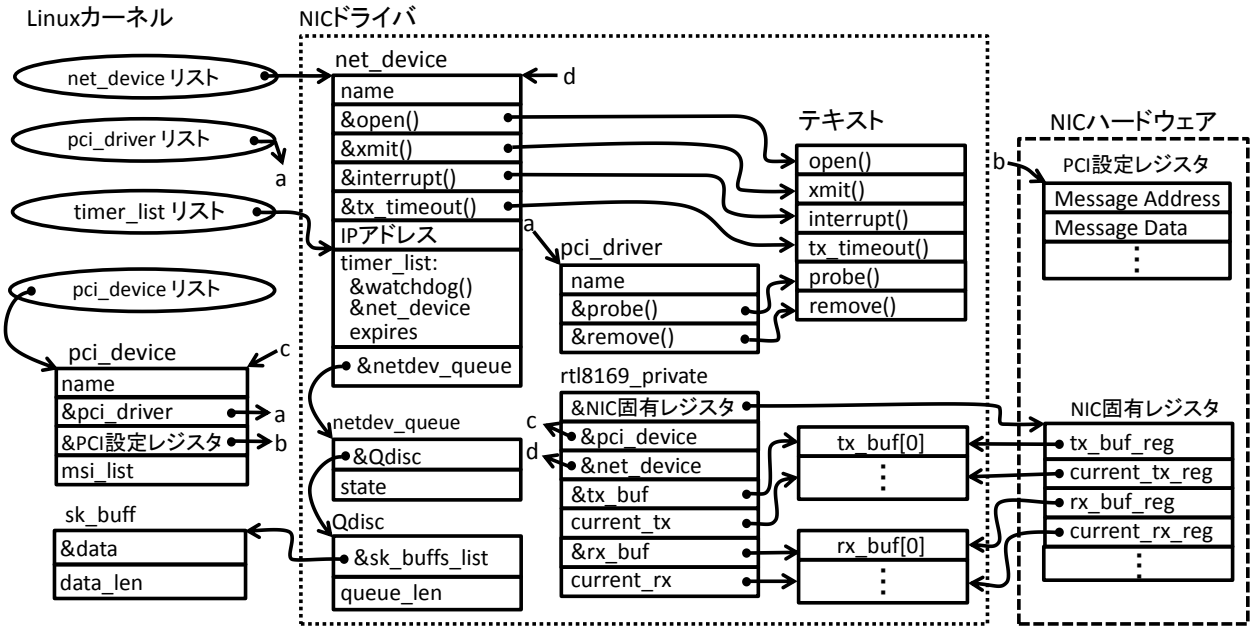

図 3.1 NIC の制御に必要な情報

(A) カーネルは NIC ドライバのテキストをメモリに配置し，NIC ドライバの登録を開始する．NIC ドライバは pci_driver を作成し，pci_driver リストに登録する．net_device と rtl8169_private はまだ作成しない．

(B) カーネルは pci_driver の probe() への参照から probe() を実行する．

(C) NIC ドライバは rtl8169_private を作成する．tx_buf と rx_buf はまだ設定しない．

(D) NIC ドライバは MSI 割込情報を設定する．msi_list，Message Address，および Message Data を初期化する．

(E) NIC ドライバは net_device を作成し，net_device リストに登録する．IP アドレスはまだ設定しない．

(F) NIC ドライバは rtl8169_private に net_device への参照を設定する．これにより，NIC と NIC ドライバを対応付ける．

(G) カーネルは pci_device に pci_driver への参照を設定する．

(3) 通信 IF の初期化

(A) カーネルは net_device の open() への参照から open() を実行する．

(B) NIC ドライバは送信キューを作成するために netdev_queue ,Qdisc を作成し，それぞれへの参照を設定する．

(C) NIC ドライバは送信バッファと受信バッファを作成する．

(D) NIC ドライバは tx_buf と rx_buf に送信バッファと受信バッファへの参照をそれぞれ設定する．

(E) NIC ドライバは tx_buf_reg と rx_buf_reg に送信バッファと受信バッファへの参照を設定する．また，NIC 固有レジスタを初期化し，NIC の通信機能をそれぞれ起動する．

(F) NIC ドライバは net_device 内の timer_list を timer_list リストへ登録する．

(G) カーネルは net_device に IP アドレスの情報を設定する．

上記の処理により，NIC の制御に必要な情報の初期化が完了する．移譲時に実行する初期化処理は，(2) と (3) であり，これらは Linux が提供するシステムコールにより実行できる．

3.5 NIC の制御を解放する処理手順

Linux が NIC の制御を解放するために実行する処理手順を以下に示す．

(1) NIC ドライバのアンロード

(A) カーネルは pci_device から pci_driver への参照を削除する．

(B) カーネルは pci_driver の remove への参照から remove を実行する．

(C) NIC ドライバは通信 IF を停止させる．このとき，timer_list を timer_list リストから削除する，

(D) NIC ドライバは NIC ハードウェアの通信機能を停止させ，tx_buf[] と rx_buf[] を削除する．また，rtl8169_private 内の tx_buf への参照と rx_buf への参照を削除する．

(E) NIC ドライバは net_device を net_device リストから外す．

(F) NIC ドライバは NIC 固有レジスタを初期化する．

(G) NIC ドライバは MSI 割込情報を削除する．msi_list，Message Address，および Message Data を初期化する．

(H) NIC ドライバは net_device を削除する．このとき，sk_buf リストに接続されている sk_buf をすべて削除する．

(I) カーネルは pci_driver リストから pci_driver を削除する．

(J) カーネルは NIC ドライバのテキスト，pci_driver，および rtl8169_private を削除する．

上記の処理により，NIC の制御に必要な情報の削除が完了する．これらは，移譲のたびに実行する必要がある．NIC ドライバのアンロードは Linux が提供するシステムコールにより実行できる．以降で，初期化処理を行うシステムコールの実行時間と初期化処理が通信処理へ与える影響を分析する．

3.6 性能測定

3.6.1 測定対象

LKM を利用した NIC 移譲では，3.4 節で述べた初期化処理を伴う．この処理は，NIC に関連するデータ構造の初期化だけでなく，ハードウェアのプローブ処理や初期化処理を含むため，ソフトウェア処理のみに比べて時間がかかると予想される．ここでは，各処理の時間が移譲処理時間に与える影響の度合を測定対象とする．また，移譲処理時間に対して実際の通信停止時間がどう関係するかについても明らかにする．したがって，初期化処理を行うシステムコールの実行時間と移譲によって発生する通信停止時間を測定対象とする．

システムコールの実行時間は，NIC ドライバのアンロード，ロード，および通信 IF の初期化を行うシステムコールの実行時間の合計である．移譲元 OS ノードと移譲先 OS ノードの同期は考慮しない．同期時間を排除するために，移譲元と移譲先を同一 OS ノードとして，上記処理を逐次的に実行し，この時間を測定する．

通信停止時間は，LKM を利用した NIC 移譲中にそれを利用するアプリケーションプロセスがパケットを受信できない時間である．LKM を利用した NIC 移譲は NIC ドライバのロードと通信 IF の初期化により NIC の制御に必要な情報を初期化し，NIC の制御を開始する．カーネルは NIC が正常に動作することを確認した後，アプリケーションプロセスに通信機能を提供する．動作の確認は NIC からの割込を契機に実行される遅延処理によって行われる．システムコールの実行が終了した後，カーネルが NIC の動作を確認するまでアプリケーションプロセスは通信機能を利用できない．

表 3.1 移譲処理用計算機の構成

項目名	環境
OS	Mint x86_64 (Linux Kernel 3.0.8)
CPU	Intel Core i7-870 @ 2.93GHz
使用コア数	1
NIC	RTL8111/8168B PCI Express Gigabit Ethernet controller
NIC ドライバ	RTL8169

表 3.2 パケット送信用計算機の構成

項目名	環境
OS	Mint x86_64 (Linux Kernel 3.0.8)
CPU	Intel Core i7-870 @ 2.93GHz
使用コア数	4
NIC	Intel Corporation 82541PI Gigabit Ethernet controller
NIC ドライバ	e1000

測定では，2 台の計算機をネットワークで接続し，一方からパケットを連続して送信中に受信側の NIC において移譲処理を実行することで，一時的に受信不可の状態を作る．この際のシステムコールの実行時間と受信の停止時間が測定対象となる．

3.6.2 測定環境

2 台の計算機のうち，測定対象となる移譲処理計用算機 (受信側) の構成を表 5.4 に示す．同様に，パケット送信用計算機の構成を表 5.5 に示す．

システムコールの実行時間の測定には，プロセッサの動作クロックに合わせて加算されるクロックカウンタを使用した．また，測定誤差を最小とするため，すべての計測はシングルユーザモードで実行した．

3.6.3 測定手順

測定手順を以下に示す.

(1) パケット送信用計算機からパケット受信用計算機へ 1 ms 間隔で 1 KB の UDP パケットを送信し続ける．データ部先頭は，パケット番号として連番を挿入しておく．パケットは合計で 10,000 個送信する．

(2) 移譲処理用計算機でパケットを受信し続け，受信したパケット番号を記録する．

(3) パケットを 5,000 個受信して通信の安定性を確認した後，移譲処理用計算機上で NIC ドライバのアンロード，ロード，および通信 IF の初期化を行うシステムコールを逐次実行する．

上記のうち，測定対象で述べた「システムコールの実行時間」は，(3) の処理にかかる時間として測定する．また,「通信停止時間」は，受信したパケット番号の記録から欠落したパケット数を算出し，パケット送信間隔の 1 ms を乗じてその時間とする．

3.6.4 測定結果

システムコールの実行時間を表 3.3 に示し，システムコールの実行時間と通信停止時間の関係を表 3.4 に示す．表 3.3 から，システムコールの実行時間の合計は，57.69 ms であると分かる．そのうちの 93.8 % である 54.15 ms は，通信 IF の初期化の時間に費されていることが分かる．通信 IF の初期化は NIC ハードウェアに対して I/O 命令を発行することで，レジスタの初期化を行う．この I/O 命令がシステムコールの実行時間の大半を占めることが分かった．

また，表 3.4 から，通信停止時間は，システムコールの実行時間の 57 ms に比較して約 40 倍の 2,259 ms であった．これは，システムコールの実行時間に比べて 2,202 ms 長い．これは，LKM を利用した移譲処理に関係する一連の処理，NIC ドライバのアンロード，ロード，および通信 IF の初期化の処理の完了後も，約 2 s 間は，NIC を使用できないことを意味する．

3.6.5 通信停止中の処理内訳

「システムコールの実行時間」と「通信停止時間」の間に大きな差が発生する原因について，通信停止中の処理時間の内訳を図 3.2 に示して，以下で説明する．

表 3.3 システムコールの実行時間

処理名	時間 (ms)	割合 (%)
NIC ドライバのアンロード	2.27	4.0
NIC ドライバのロード	1.27	2.2
通信 IF の初期化	54.15	93.8
合計	57.69	

表 3.4 システムコールの実行時間と通信停止時間の関係

項目名	時間 (ms)
システムコールの実行時間	57
通信停止時間	2259
差分	2202

図中 (1) は表 3.4 中の「システムコールの実行時間」相当であり，(2) と (3) が表 3.4 中のシステムコールの実行時間と通信停止時間の差分である．

図 3.2 から，カーネルは NIC の制御に必要な情報の初期化を行うシステムコールの実行が終了した後，NIC のリンクアップを検出するまでの 2,202 ms 間アプリケーションに通信機能を提供しないことが分かる．

通信 IF の初期化からカーネルがアプリケーションに対して通信機能を提供するまでの処理流れを以下で説明する．通信 IF の初期化時，NIC ドライバは NIC ハードウェアを通信デバイスとして利用するために NIC 固有レジスタを初期化 (3-D) する．(3-D) を契機として NIC ハードウェアの通信機能が起動 (4) する．NIC ハードウェアは計算機外部と通信可能であること (リンクアップ) を確認したのち割込を通知する (5-A)．カーネルはこの割込を受けて NIC ハードウェアのリンクアップを検出 (6) し，アプリケーションに対して通信機能の提供を開始する (7)．

上記の処理のうち (4) と (5) にかかる時間は通信停止時間の約 100 %である 2,202 ms である．一方，(6) と (7) の合計処理時間は約 20 μ s と極めて短い．以上から，システムコールの実行時間と通信停止時間の間に大きな差が発生する原因は，NIC ハードウェアのリンクアップを確認するための待ち時間であることがわかる．これは，NIC ハードウェア内の処理であるため，通信停止時間を短縮することは難しいと考えられる．このことから，LKM を利用

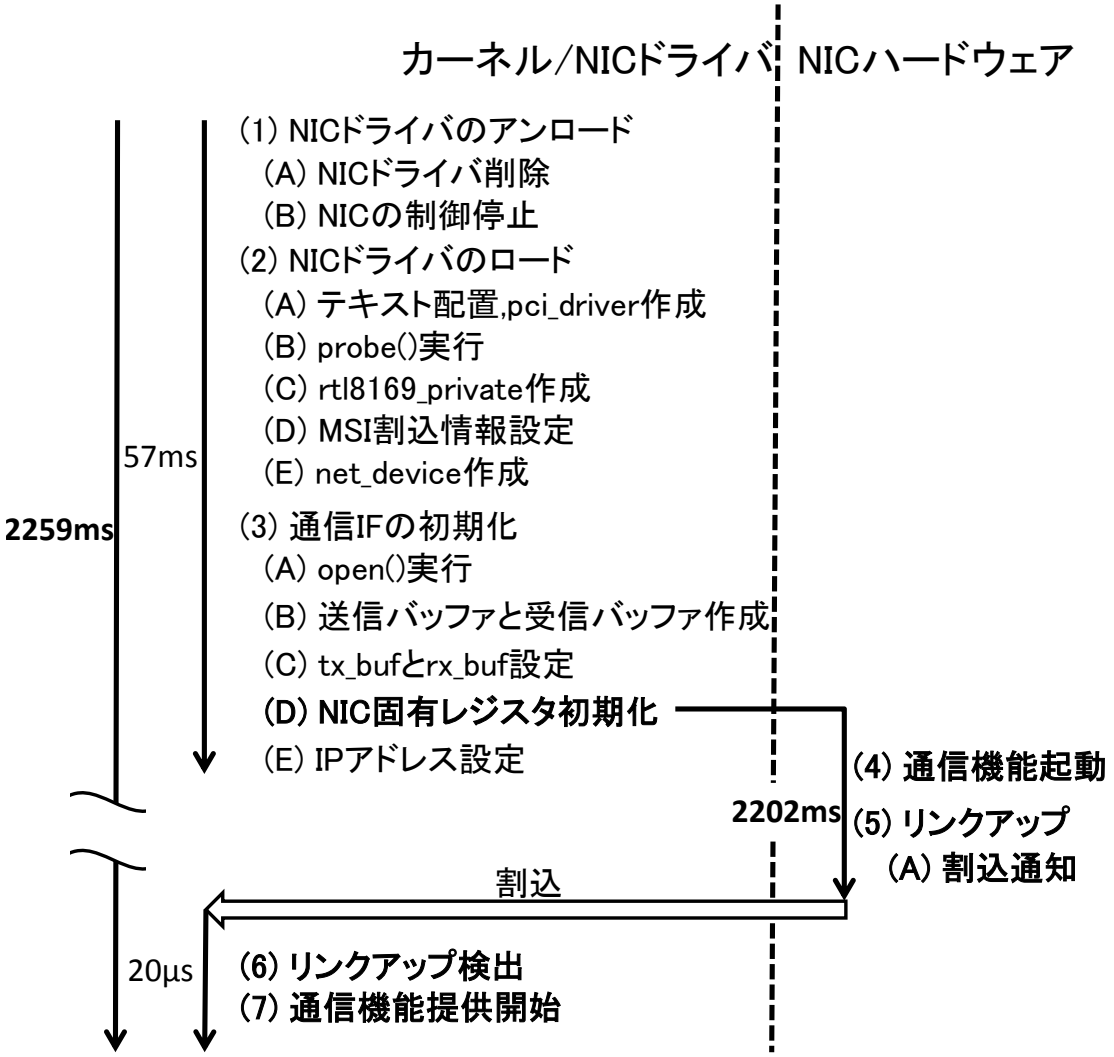

図 3.2 通信停止中の処理内訳

した移譲方式は短い間隔で移譲を繰り返す用途，すなわち NIC の時分割に適用することは難しい.

第 4 章

割込ルーティングの変更による移譲方式

4.1 更新する情報の削減による移譲処理の高速化

3 章で述べた LKM を利用した移譲方式は，移譲元と移譲先の双方とも自身が持つ NIC の制御に必要な情報を全て移譲時に初期化する．短い間隔で移譲を繰り返す状況，すなわち NIC を時分割することを考えた場合，この処理には無駄があるといえる．具体的には，I/O ポートアドレスのような NIC に関する基本的な情報や，初期化の過程で設定した IP アドレスといった情報は，一度設定すれば，変化しない情報である．このため，各 OS ノードはかならず 1 回は初期化処理を必要とするものの，2 回目以降に移譲を受ける場合には，以前の初期化で得た情報を保留しておき，その情報を再利用することで，移譲が完了する筈である．

しかしながら，すべての情報を保留して再利用することは困難である．例えば，NIC ハードウェアの情報は NIC を占有する OS ノードによって変化する．具体的には，NIC ハードウェアの参照する送受信バッファのアドレスは，OS ノードによって変化するため，移譲時に更新が必要である．また，NIC ハードウェアの情報の変化に合わせて，NIC ドライバやカーネルの情報を更新しなければならない．

このことから，NIC ドライバや NIC ハードウェアが持つデータ構造とレジスタは，移譲を経ても保留可能な領域と，移譲を受ける際に更新が必要な領域とに大別できるといえる．移譲を受ける際に更新が必要な情報のみを処理対象とすることで，高速な移譲処理が実現できると考えられる．

以降で，図 3.1 で示したデータ構造とレジスタから，移譲を受ける際に更新が必要なものを選別する．以下の 2 つは NIC ハードウェアのレジスタであり，移譲先の OS ノードによって上書きされる．つまり，NIC の制御を取得する際に更新が必要であると考えられる．

(1) PCI 設定レジスタ

(2) NIC 固有レジスタ

また，以下は NIC ドライバがメモリ上に保持するデータ構造であり，移譲先の OS ノードによって上書きされない．

(3) `rtl8169_private`

しかし，送信バッファと受信バッファを操作するための情報を持つ．NIC ドライバと NIC ハードウェアは送信バッファと受信バッファについて，互いに同じエントリを参照していることを前提として動作する．よって，NIC の制御を取得する際に NIC ハードウェアのレジスタを更新した場合，使用するバッファのエントリを同期するために NIC ハードウェアのレジスタの変化に追従するために更新が必要であると考えられる．

一方，図 3.1 で示した情報のうち，これら 3 つ以外の情報に関しては，カーネルと NIC ドライバがメモリ上に確保するデータ構造であり移譲先の OS ノードによって変更されない．また，NIC の変更に追従する必要がない．よって，2 回目以降に NIC の制御を取得する場合には，更新なしで再利用できる．

ここで，更新の必要があると述べた (1)，(2)，および (3) に関しても，すべてを更新する必要はない．以上から，(1)，(2)，および (3) で管理される情報の中から更新が必要な情報を特定し，部分的に更新することで NIC を移譲できると考えられる．

以降で，パケットの送信処理と受信処理を分析し，NIC の制御を取得する際に更新が必要な情報を特定する．

4.2　NIC の制御取得時に更新する必要のある情報の特定

4.2.1　送信処理と受信処理の処理流れ

図 4.1 に送信処理と受信処理の処理流れを示し，以下で説明する．

(1) 送信処理

(A) カーネルは送信処理に使用したい通信デバイスの `net_device` を参照し，`xmit()` を実行する．

(B) `net_device` の `xmit()` への参照から，NIC ドライバの `xmit()` は実行される．

(C) NIC ドライバは xmit() の引数として受け取ったソケットバッファからパケットを取得し，送信バッファにパケットを格納する．パケットを格納する送信バッファのエントリは，rtl8169_private 内の current_tx の値で決定する．パケット格納後，NIC ドライバは current_tx を次のエントリを指すよう更新する．送信バッファの末尾を指していた場合，NIC ドライバは次に送信バッファの先頭を指すように tx_buf の値で current_tx を初期化する．

(D) NIC は送信バッファからパケットを取得し，送信する．パケットを取得する送信バッファのエントリは，current_tx_reg の値で決定する．パケット取得後，NIC は current_tx_reg を次のエントリを指すよう更新する．送信バッファの末尾を指していた場合，NIC は次に送信バッファの先頭を指すように tx_buf_reg で current_tx_reg を初期化する．

(E) NIC はパケットを計算機外部へ送信する．

(F) NIC はパケットの送信が完了したことを通知するために MSI で割込を発行する．具体的には，Message Address に格納されているアドレスへ Message Data に格納されている値を書き込むことで割込を通知する．

(2) 受信処理

(A) NIC は計算機外部からパケットを受信する．

(B) NIC は受信バッファにパケットを格納する．パケットを格納する受信バッファのエントリは，current_rx_reg の値で決定する．パケット格納後，NIC は current_rx_reg を次のエントリを指すよう更新する．受信バッファの末尾を指していた場合，NIC は次に受信バッファの先頭を指すように rx_buf_reg の値で current_rx_reg を初期化する．

(C) NIC はパケットの受信が完了したことを通知するために MSI で割込を発行する．具体的には，Message Address に格納されているアドレスへ Message Data に格納されている値を書き込むことで割込を通知する．

(D) NIC ドライバは受信バッファからパケットを取得する．パケットを取得する受信バッファのエントリは，rtl8169_private 内の current_rx の値で決定する．パケット取得後，NIC ドライバは current_rx を次のエントリを指すよう更新する．受信バッファの末尾を指していた場合，NIC ドライバは次に受信バッファの先頭を指すように rx_buf の値で current_rx を初期化する．

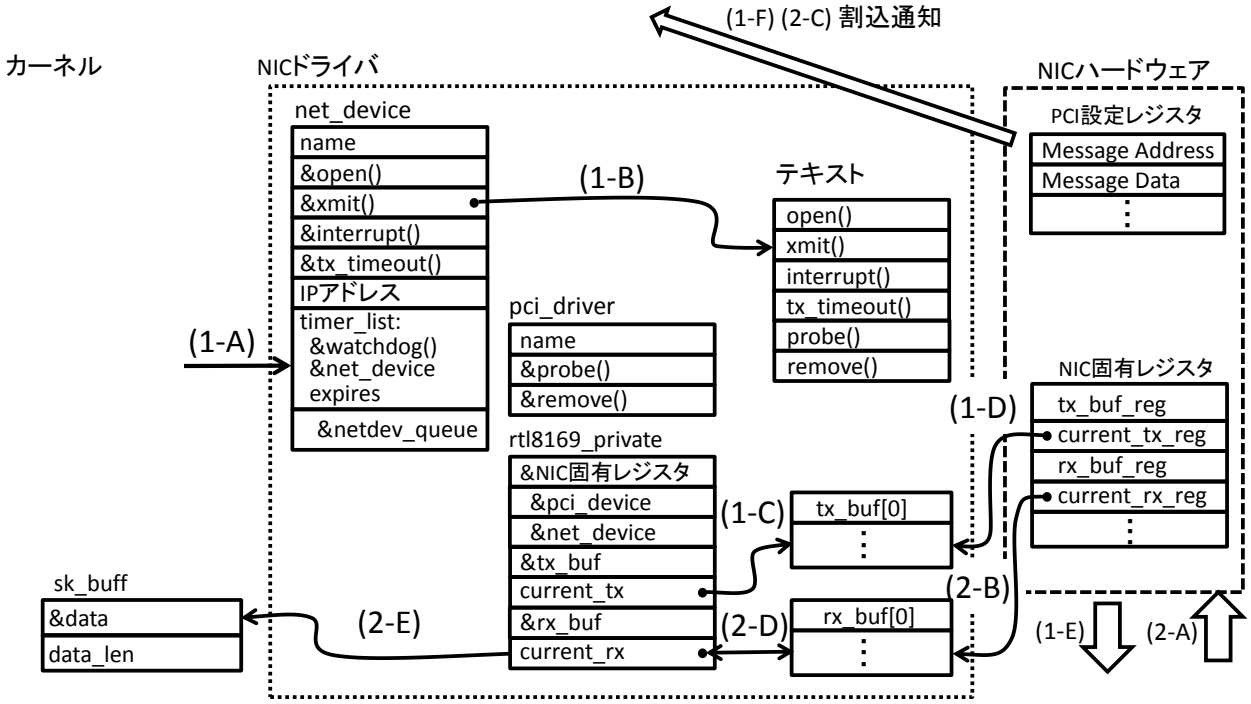

図　4.1　送信処理と受信処理の処理流れ

(E) NIC ドライバはソケットバッファを介して，カーネルに受信したパケットを渡す.

4.2.2　NIC の制御取得時に更新が必要な情報

4.2.1 項から，割込設定と PCI 設定レジスタ，NIC 固有レジスタ，および `rtl8169_private` が持つ情報のうち，NIC を移譲する際に更新が必要な情報は以下の 8 つである.

(1) `Message Address`

(2) `Message Data`

(3) `tx_buf_reg`

(4) `current_tx_reg`

(5) `rx_buf_reg`

(6) `current_rx_reg`

(7) current_tx

(8) current_rx

(1) と (2) は NIC が MSI で割込通知する際に使用する．(3)〜(6) は NIC が送信バッファと受信バッファにアクセスするために使用する．(7) と (8) は NIC ドライバが送信バッファと受信バッファにアクセスするために使用する．

上記の 8 つの情報を更新することで NIC がパケットの送信処理と受信処理の対象とする OS ノードを切り替えることができる．また，他の情報は，NIC を移譲する際に変更されることのない情報であり，保留し続けることができる．なお，rtl8169_private の tx_buf と rx_buf への参照は，パケットの送受信処理に使用する情報であるが NIC の情報の更新に追従する情報ではないため，更新する必要はない．

4.2.3 NIC の制御を取得する処理手順

割込ルーティングの変更による NIC の制御取得手順を以下に示す．前提条件として，NIC ドライバのロードと通信 IF の初期化を完了しているものとする．これは，LKM を利用した移譲の場合と変わらない．

(1) **割込ルーティングの変更**
PCI 設定レジスタは，MSI の割込情報を以下の 2 つのレジスタに持つ．

Message Data: 書き込むメッセージの内容を保持する．割込ベクタ番号を指定する．

Message Address: メッセージを書き込むアドレスを保持する．割込通知先コア ID を指定する．

上記のレジスタ値は，pci_device の msi_list で管理されている．移譲先 OS ノードは msi_list から取得した情報を 2 つのレジスタに書き込み，割込ルーティングを自身のものに復元する．

(2) **NIC が参照するバッファの変更**
NIC は NIC 固有レジスタの current_tx_reg と current_rx_reg で使用するバッファのエントリを決定する．NIC ハードウェアの仕様上，これらのレジスタは I/O 命令で直接更新できない．そこで，ソフトリセットという NIC ハードウェアの機能を利用して更新する．ソフトリセットは NIC ハードウェアの再起動無しで NIC ハードウェア内のパケット送受信に関する情報を初期化する．ソフトリセットにより，NIC ハードウェ

アは current_tx_reg と current_rx_reg の値をそれぞれ tx_buf_reg と rx_buf_reg の値で初期化する．ここで，tx_buf_reg と rx_buf_reg は I/O 命令により更新できる．よって，tx_buf_reg と rx_buf_reg を変更し，ソフトリセットを実行することで NIC ハードウェアが参照するバッファを変更する．

(3) **NIC ドライバと NIC が参照するバッファのエントリの同期**
(2) により current_tx_reg と current_rx_reg は，移譲先 OS ノードの送信バッファと受信バッファの先頭を指す．一方，移譲先 OS ノードの current_tx と current_rx は以前の情報を保留している．ここで，NIC ドライバと NIC ハードウェアが参照する送信バッファと受信バッファのエントリを同期する必要がある．よって，current_tx と current_rx が送信バッファと受信バッファの先頭を指すように同期する．

上記の 3 つの処理により，移譲を受ける際に更新が必要な情報のみを初期化し，NIC の制御を取得できる．

4.3 NIC の利用を停止する処理

4.3.1 NIC を非占有の OS ノードにおける問題

4.2 節で NIC の制御を取得する際に更新が必要な情報を特定した．また，これらの情報のみを初期化し，NIC の制御を取得する手順を示した．割込ルーティングの変更による移譲方式は，LKM を利用した移譲方式に比べて移譲時に初期化する情報量が少ない．また，NIC ハードウェアの再起動なしで移譲できるため，移譲にともなう通信停止時間を大幅に削減できる．

一方で，割込ルーティングの変更による移譲方式では，NIC の占有/非占有に関わらず，すべての OS ノードは NIC ドライバのロードと通信 IF の初期化を完了している必要がある．これにより，NIC を非占有の OS ノードにおいて以下の問題が発生する．

(問題 1) 非占有の NIC のレジスタの操作
Linux では，カーネルは NIC ドライバを介して NIC のレジスタへアクセスする．割込ルーティングの変更による移譲方式では，すべての OS ノードが NIC ドライバをロードする必要がある．このため，NIC を占有していない OS ノードも NIC ドライバを介して NIC のレジスタを操作できる．これにより，NIC を非占有の OS ノードがカーネルの終了や NIC ドライバのアンロードを契機として，NIC のレジスタを初期化してしまう可能性がある．

(問題 2) ウォッチドッグによる NIC ドライバの再起動

ウォッチドッグはカーネルの機能であり，NIC ドライバの状態を監視する．NIC ドライバの送信処理が一定時間正常に実行されなかった場合，ウォッチドッグはこれを検出し NIC ドライバと NIC ハードウェアを再起動する．NIC を非占有の OS 上でパケット送信処理を実行するとパケットの送信は失敗する．ウォッチドッグはこれを検出し，NIC ドライバと NIC ハードウェアを再起動してしまう．

(問題 3) 非占有の NIC を利用可能と誤認

カーネルは NIC ドライバを通信プロトコルスタックへ登録することにより NIC を通信 IF として認識する．割込ルーティングの変更による移譲方式では NIC ドライバをアンロードすることなく NIC を移譲する．このため，カーネルは非占有である NIC を利用可能通信 IF な誤認し，パケット送信を依頼する．しかしながら，NIC は非占有であるため，パケットは送信されることなく NIC ドライバで破棄される．カーネルは NIC ドライバへパケットを渡した時点でプロセスへ送信成功を返すため，パケットは送信されることなく破棄されているにも関わらず，プロセスは送信に成功したと誤認する問題が発生する．

以降で上記の問題に対処し，割込ルーティングの変更による移譲方式における NIC の利用を停止する処理を実現する．

4.3.2 対処

4.3.1 項で示した問題への対処を以下に示す．

(対処 1) NIC のレジスタへの書き込みを禁止

(問題 1) への対処である．

Linux における NIC のレジスタへの書き込みはすべて NIC ドライバを介して行われる．そこで，NIC ドライバを改変し，NIC を非占有時は NIC のレジスタへの書き込みを禁止する．これにより，NIC のレジスタへ書き込み可能な OS を NIC を占有している OS ノードに限定する．

(対処 2) ウォッチドッグを停止

(問題 2) への対処である．

ウォッチドッグは動的タイマという Linux の機能で実現されている．動的タイマはある処理を一定時間後に呼び出す機能である．NIC ドライバは自身の初期化処理内でウォッ

チドッグを動的タイマのリストへ登録する．カーネルはタイマ割り込みを契機として動的タイマのリストからウォッチドッグを実行する．動的タイマのリストは任意に追加，削除可能であるため，NIC 取外し間はウォッチドッグを動的タイマのリストから削除する．これにより，ウォッチドッグを停止させる．なお，ウォッチドッグの監視対象であるパケット送受信処理は，NIC を取り外している間実行されない．このため，ウォッチドッグの停止による影響を考慮する必要はない．

(対処 3) NIC ドライバへのパケット配送を禁止

(問題 3) への対処である．

カーネルから NIC ドライバへパケットを配送する処理，具体的にはパケットを送信キューへ格納する処理を禁止する．NIC ドライバへのパケット配送を禁止している様子を図 4.3 に示す．カーネルは NIC ドライバへのパケット受け渡しのインタフェースとして送信キューを持つ．本対処では，NIC を非占有の OS ノードにおいて，カーネルによる送信キューへのパケット格納を失敗させる．これにより，カーネルから NIC ドライバへのパケット配送を禁止する．この際，送信元プロセスにはエラー値を返却する．エラー値を返却することでプロセスは NIC の取外しを検出し，送信処理を停止できる．

(対処 3) により，NIC 取外し間にパケットを送信したプロセスは NIC を使用できないことを検出できる．これをを受けて送信処理を停止することで無駄な送信処理の実行を防止できる．

しかしながら，時分割のために数秒から数百ミリ秒の短い間隔で NIC の移譲を繰り返す状況を考えた場合，NIC 移譲のたびに送信処理の停止と再開を繰り返す方法は無駄が多いと考えられる．送信処理を停止したプロセスが送信処理を再開するためには，プロセス自身が送信可能か否かを判断する必要があるためである．また，NIC 取外し間に送信されたパケットはすべて NIC ドライバに渡されることなく破棄される．これにより，パケットロスが生じ，再送処理が多発すると考えられる．

上記から，短い間隔を繰り返す状況では，NIC ドライバに NIC 取外しを意識させることなく，パケットロスを最小限にする NIC 取外し手法が望ましい．そこで，短い間隔で移譲する際に有効な方法として (対処 4) を実現する．

(対処 4) パケットを NIC ドライバで保存

パケットを NIC ドライバで保存する様子を図 4.2 に示す．NIC ドライバは，カーネルから受け取ったパケットを NIC ハードウェアへ渡さず，NIC ドライバ内に用意したキュー

へ格納する．これにより，NIC を非占有時に NIC ドライバへ渡されたパケットを破棄することなく保存できる．なお，このとき NIC ドライバはカーネルへ送信成功の値を返却する．これにより，プロセスとカーネルはパケットの送信に成功したと認識する．キューへ保存したパケットは，NIC 取得時にまとめて送信する．

(対処 4) により，プロセスへ NIC の取外しを意識させることなく，NIC ドライバのパケット送信処理実行を防ぐことができる．また，NIC ドライバ内でパケットを保存することでパケットロスを防ぎ，再送処理の発生回数の削減が期待できる．プロセスは NIC の取外しを意識しないため，NIC の取外し/取付けごとに送信処理を停止/再開する必要がなく，短時間で移譲を繰り返す際の送信処理のオーバヘッドを削減できる．しかしながら，(対処 4) の方法は NIC ドライバ内に用意できるキューの容量に限界があり，長時間 NIC を取得できない場合に対応できないという問題がある．

そこで，(対処 3) と (対処 4) を併用することで，短時間で移譲を繰り返す際の通信処理のオーバヘッドを削減しつつ，長時間の取外しにも対応する．具体的には，NIC 取外し直後は (対処 4) により，プロセスへ NIC の取外しを意識させずパケットを NIC ドライバ内のキューへ保存する．長時間 NIC が取り外され，キューが満杯になってしまった後は (対処 3) により，NIC ドライバへのパケット渡しを禁止し，プロセスへ NIC の取外しを通知する．

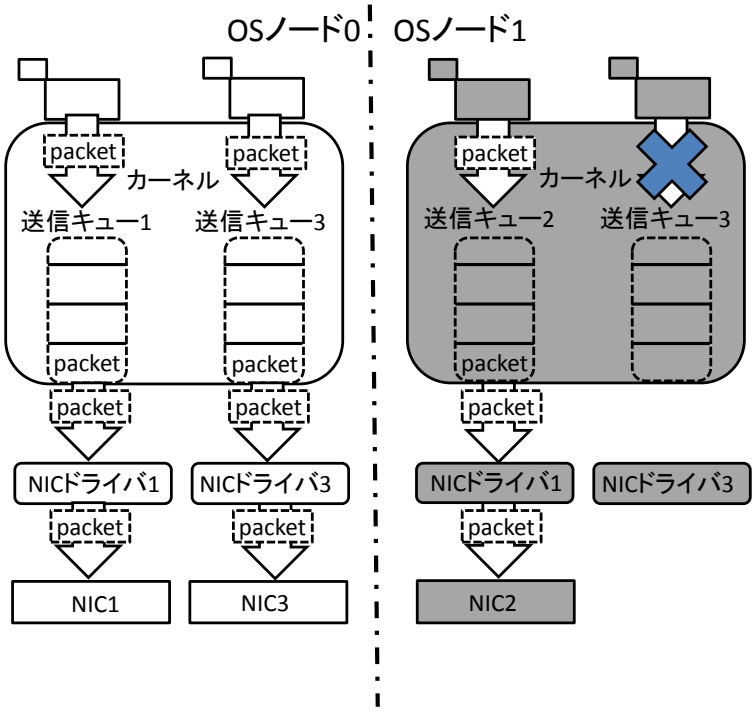

図　4.2　NIC ドライバへのパケット渡しを禁止する方法

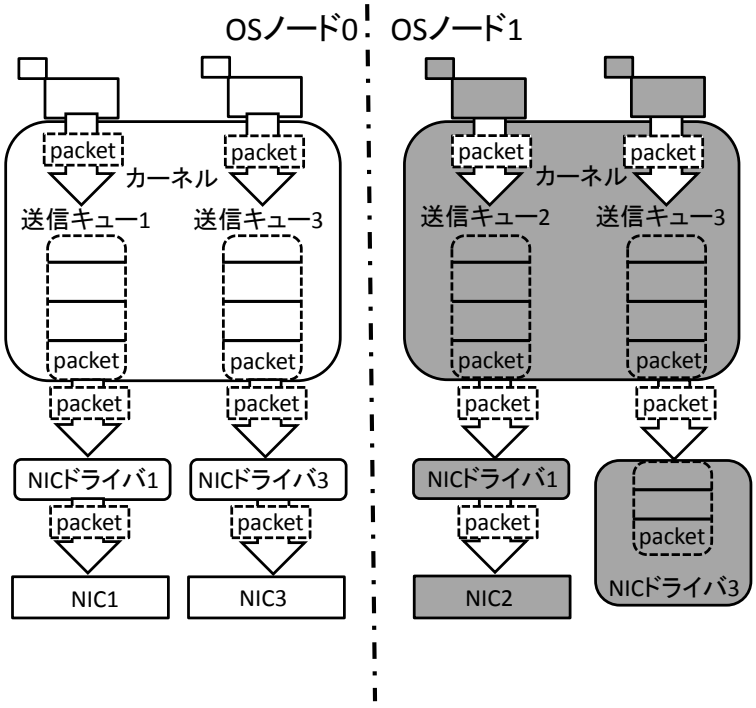

図　4.3　パケットを NIC ドライバで保存する方法

第 5 章

評価

5.1 移譲時に初期化する情報量の比較

LKM を利用した移譲方式と割込ルーティングの変更による移譲方式について，初期化する情報を示し，比較する．

各移譲方式について初期化する情報を表 5.1，表 5.2，および表 5.3 に示し，以下で説明する．なお，表中の○は手を加えない情報，△は一部初期化する情報，×はすべて削除もしくは作成する情報である．

(1) LKM を利用した移譲方式

LKM を利用し NIC ドライバを動的に挿抜する．NIC ドライバのロード時の初期化を利用して NIC を移譲する．NIC ドライバを挿抜するため，移譲ごとに NIC ドライバが保持する情報はすべて削除されて作り直される．また，NIC ハードウェア内のレジスタは NIC ドライバアンロード時にすべて削除され，ロード時に初期化される．NIC ドライバのロード後は，NIC ドライバを通信 IF として再度登録する必要がある．一方で，カーネルの保持する情報については，リストへの追加削除を除くとほとんど手を加えない．

(2) 割込みルーティングの変更による移譲方式

各 OS ノードは，NIC の制御に必要な情報を保持したまま最小限の変更によって，制御を移譲する．NIC を占有しない OS ノードも NIC ドライバをロードし，通信 IF を初期化している．このため，NIC を占有していない OS ノードも NIC を利用可能な通信 IF として認識している．表 5.1 について，NIC 非占有時にウォッチドッグを停止させ

表 5.1 NIC 移譲時に初期化する情報 (カーネル)

名前	概要	LKM		割込ルーティング	
		取外し	取付け	取外し	取付け
pci_device リスト	PCI デバイスの情報を管理するデータ構造 pci_device のリストである.	○	○	○	○
sk_buff	パケットを格納するためのデータ構造である.	×	×	○	○
pci_device	PCI デバイスの情報を格納するためのデータ構造である.	△	△	○	○
pci_driver リスト	PCI ドライバの情報を管理するデータ構造 pci_driver のリストである.	△	△	○	○
net_device リスト	通信 IF の情報を管理するデータ構造 net_device のリストである.	△	△	○	○
timer_list リスト	動的タイマを管理するデータ構造 timer_list のリストある.	△	△	○	○

るため，timer_list からウォッチドッグに対応する timer_list を削除する．NIC ドライバと NIC ハードウェアの保持する情報については，4.2.2 項で示した情報を初期化する．

上記からカーネル，NIC ドライバ，および NIC ハードウェアのすべてにおいて，割込ルーティングの変更による移譲方式は LKM を利用した移譲方式と比べて，初期化する情報を大幅に削減できていることが分かる．LKM を利用した移譲方式は，NIC ドライバのアンロードに伴い NIC ドライバと NIC ハードウェアのすべてを初期化する．とくに NIC ハードウェアの初期化は，リンクアップ待ち時間を発生させ，2 秒ほどの通信停止時間を発生させる．

一方で，割込ルーティングの変更による移譲方式は，NIC ドライバと NIC ハードウェアの情報をほとんどを保持する．また，初期化する場合も一部の変数とレジスタに限定し，初期化するデータ量を最小限にしている．このため，割込ルーティングの変更による移譲方式は LKM を利用した移譲方式と比較して，高速な移譲ができると期待できる．以降で，割込ルーティングの変更による NIC 移譲を行った際の通信停止時間を測定し，LKM を利用した移譲方式と比較する．

表 5.2 NIC 移譲時に初期化する情報 (NIC ドライバ)

名前	概要	LKM		割込ルーティング	
		取外し	取付け	取外し	取付け
pci_driver	PCI ドライバの情報を管理するデータ構造である.	×	×	○	○
net_device	通信 IF の情報を管理するデータ構造である.	×	×	○	○
Qdisc	送信キューの実体である. ソケットバッファのリストの先頭を持つ.	×	×	○	○
テキスト	NIC ドライバの管理する静的なデータである.	×	×	○	○
rtl8169_private	NIC ハードウェアの操作に必要な情報を管理するデータ構造である.	×	×	○	○
tx_buff	NIC ドライバと NIC ハードウェア間で送信パケットを授受する際に使用する配列である.	×	×	△	△
rx_buff	NIC ドライバと NIC ハードウェア間で受信パケットを授受する際に使用する配列である.	×	×	△	△
netdev_queue	送信キューを管理するデータ構造である.	×	×	○	○

5.2 移譲処理が通信処理に与える影響の比較

5.2.1 測定対象

LKM を利用した移譲と割込ルーティングの変更による移譲は 5.1 節で示した情報をそれぞれ初期化する. 割込ルーティングの変更による移譲では, 移譲時に NIC ドライバと NIC ハードウェアの持つ情報の大半を初期化することなく保留し, 使用する. とくに, NIC ハードウェアの内部状態を初期化しないため, NIC ハードウェアの再起動が発生しない. これにより, LKM を利用した移譲における通信停止時間のおもな原因であるリンクアップ待ちなしで移譲できるため, 高速であると予想される.

ここでは, 各 NIC 移譲処理が Linux の通信処理に与える影響の度合いを測定対象とする. 具体的には, 移譲によって発生する通信停止時間を測定対象とする. 通信停止時間は NIC 移

表 5.3 NIC 移譲時に初期化する情報 (NIC ハードウェア)

名前	概要	LKM		割込ルーティング	
		取外し	取付け	取外し	取付け
PCI 設定レジスタ	PCI デバイスで共有の情報を管理するレジスタである．MSI で使用する情報を持つ．	×	×	△	△
NIC 固有レジスタ	NIC ハードウェアの制御に必要な情報を管理するレジスタである．送受信バッファのアクセスに必要な情報を持つ．	×	×	△	△

表 5.4 移譲処理用計算機の構成

項目名	環境
OS	Mint x86_64 (Linux Kernel 3.0.8)
CPU	Intel Core i7-870 @ 2.93GHz
使用コア数	移譲元 OS ノード: 2 個，移譲先 OS ノード: 2 個
NIC	RTL8111/8168B PCI Express Gigabit Ethernet controller
NIC ドライバ	RTL8169

譲中にそれを利用するアプリケーションプロセスがパケットを送信できない時間である．

測定では，2 台の計算機をネットワークで接続し，一方からパケットを連続して送信中に送信側の NIC において移譲処理を実行することで，一時的に送信不可の状態を作る．この際の送信の停止時間が測定対象となる．

5.2.2 測定環境

測定に使用する計算機のうち，測定対象となる移譲処理計用算機 (送信側) の構成を表 5.4 に示し，同様に，パケット受信用計算機の構成を表 5.5 に示す．

表 5.5 パケット受信用計算機の構成

項目名	環境
OS	Mint x86_64 (Linux Kernel 3.0.8)
CPU	Intel Core i7-870 @ 2.93GHz
使用コア数	4
NIC	Intel Corporation 82541PI Gigabit Ethernet controller
NIC ドライバ	e1000

5.2.3 測定手順

測定手順を以下に示す.

(1) 移譲処理用計算機上で移譲元 OS ノードと移譲先 OS ノードを同時走行させる.

(2) 移譲元 OS ノードで NIC を取得する.

(3) 移譲元 OS ノードと移譲先 OS ノードはパケット受信用計算機へのパケット送信を同時に開始する.

(4) 移譲元 OS ノードと移譲先 OS ノードからパケット受信用計算機へ 1 ms 間隔で 1 KB の UDP パケットを送信し続ける．データ部先頭は，パケット番号として連番を挿入しておく.

(5) 移譲処理用計算機でパケットを受信し続け，受信したパケット番号を記録する.

(6) パケットを 5,000 個受信して通信の安定性を確認した後，移譲元 OS ノードは NIC を取り外す．NIC の取外しが完了した後，移譲先 OS ノードへ NIC 取外しの完了を通知する．移譲先 OS ノードは移譲元 OS ノードから NIC の取外しを受け取った後，NIC の取付を実行する.

(7) パケット受信用計算機はパケットを 10,000 個受信した時点で終了する.

上記のうち，また,「通信停止時間」は，5,000 個目のパケットを受信した後 5,001 個目のパケットを受信するまでの時間とする．つまり移譲元 OS ノードが NIC を取り外してから移譲先 OS ノードが NIC を取得しパケット送信を再開するまでの時間である.

なお，NIC を占有していない OS ノードから送信したパケットは送信に失敗する．このとき，LKM を利用した移譲ではパケットはカーネルによって破棄される．割込ルーティングを利用した移譲では，NIC ドライバに用意したキューに空きがある場合は，キューに格納される．キューに空きがない場合は，カーネルによって破棄される．

5.2.4 測定結果

LKM を利用した移譲が通信処理に与える影響を図 5.1 に示し，割込ルーティングの変更による移譲が通信処理に与える影響を図 5.2 に示す．また，各移譲方式における通信停止時間を表 5.6 に示す．図 5.1 から，5,000 個目のパケットを受信してから，次のパケットを受信するまで通信が途切れていることが分かる．これは，NIC 移譲によって通信が停止しているためである．表 5.6 から，LKM を利用した NIC 移譲における通信停止時間は 2,932 ms である．一方で，割込ルーティングの変更による移譲では，通信停止時間は 0.51 ms であり，LKM を利用した移譲と比較して，非常に高速に移譲できていることがわかる．

図 5.2 において，5,000 個以降にパケット 3,000 個を即座に受信し，その後 2,000 個のパケットを NIC 移譲前と同じ間隔で受信している．これについて，図 5.3 で示し，説明する．本測定では，移譲元 OS ノードと移譲先 OS ノードは同時にパケット送信を開始する．このとき，移譲元 OS ノードの送信したパケットは受信側に到着する．一方，移譲先 OS ノードで送信したパケットは，まず NIC ドライバ内のキューへ格納される．キューの容量は 3,000 個のため，移譲先 OS ノードの送信する 5,000 パケットのうち，3,000 個はキューに保存される．キューが満杯になった後，NIC ドライバへのパケット配送が禁止される．これにより，残りの 2,000 個は送信に失敗する．NIC 取得時，移譲先 OS ノードはまずキューに格納しているパケットをすべて送信する．これにより，受信側は 3,000 個のパケットを一気に受信する．その後，NIC 取得後に送信したパケットを 2,000 個受信し，終了する．

測定結果から，割込ルーティングの変更による移譲は 1 ms 未満の通信停止時間で NIC を移譲できる．また，本来破棄されるはずのパケットをキューへ保存することで NIC 取外し間におけるパケットロスを削減できることがわかる．このことから，割込ルーティングの変更による移譲は数秒から数百ミリ秒間隔で移譲が発生する NIC の時分割へ適用可能であると考えられる．

表 5.6 NIC 移譲による通信停止時間

移譲方式	通信停止時間
LKM を利用した NIC 移譲	2,932ms
割込ルーティングを利用した NIC 移譲	0.51ms
差分	2,931ms

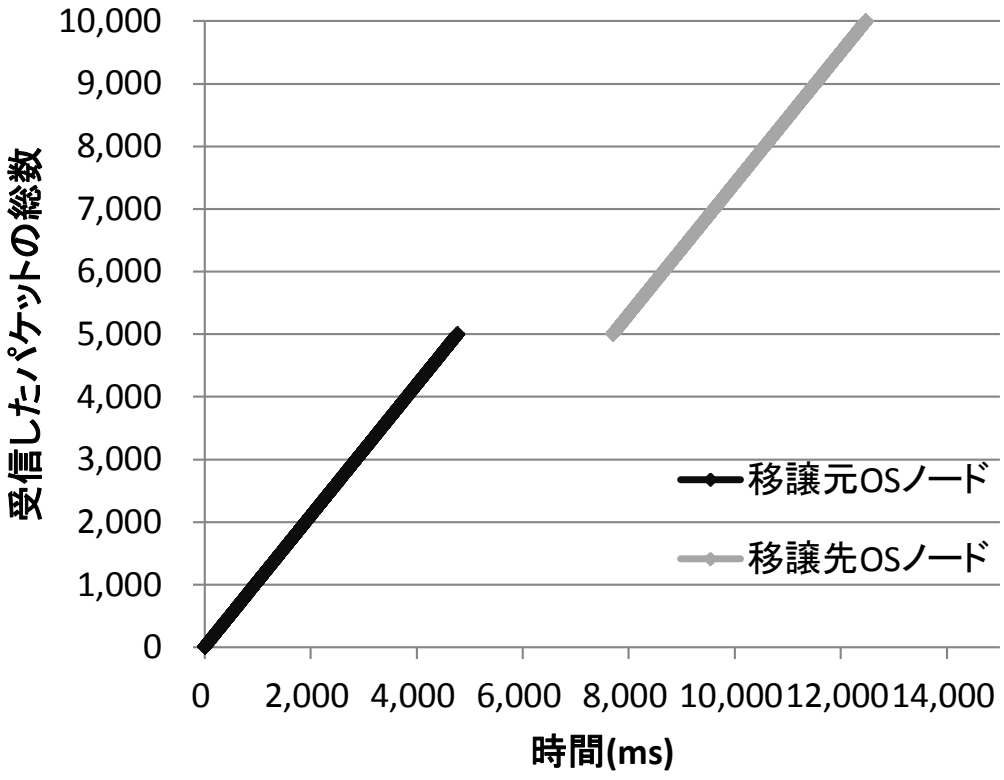

図 5.1 LKM を利用した移譲が通信処理に与える影響

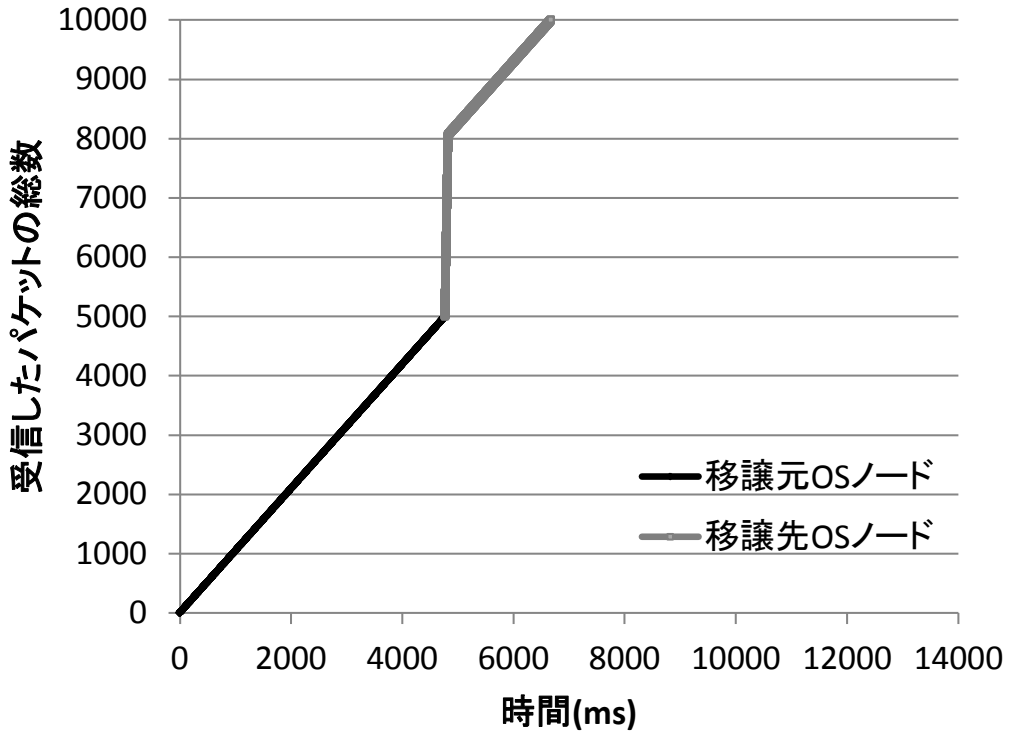

図　5.2　割込ルーティングの変更による移譲が通信処理に与える影響

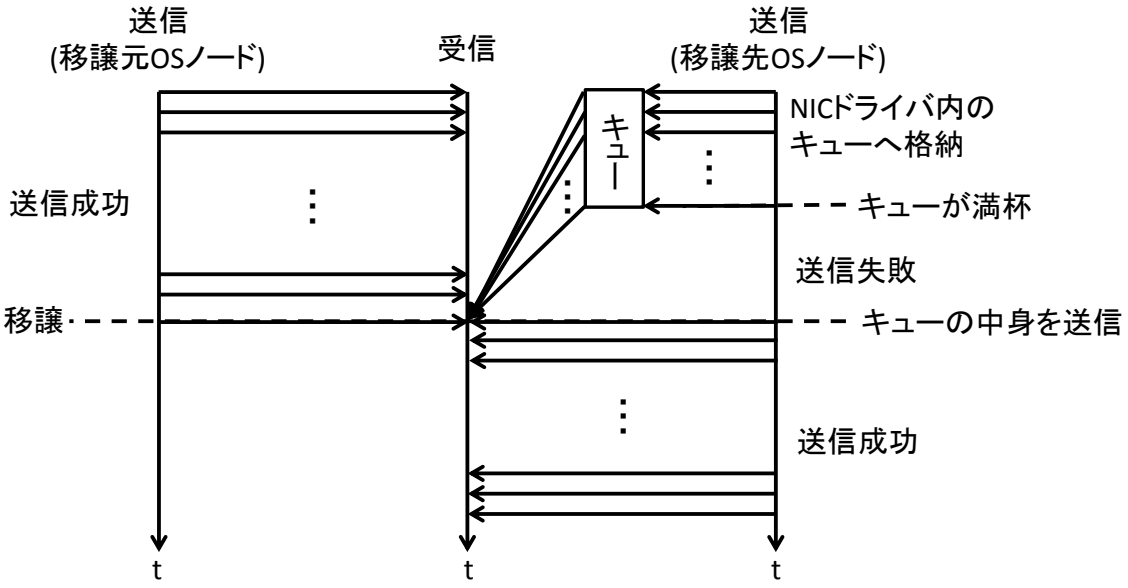

図　5.3　キューを利用したパケット送信処理

第 6 章

おわりに

Mint において，1 つの NIC を複数 OS で利用する方法として，NIC の時分割を提案した．NIC の時分割を実現するためには，NIC を利用する OS ノードを動作中に切り替える必要がある．そこで，割込ルーティングの変更による移譲方式を提案し，LKM を利用した移譲方式と比較した．

LKM を利用した移譲方式は，Linux の標準機能で実行でき，幅広い I/O デバイスに適用可能である．しかし，移譲のたびに NIC ドライバと NIC ハードウェアが持つ情報をすべて初期化する必要がある．移譲処理に発生する通信停止時間を測定した結果，通信 IF の初期化により NIC ハードウェアのレジスタが初期化されることで，リンクアップ待ちが発生し約 2 s 間通信が停止することが分かった．NIC を時分割で利用する場合，数百ミリ秒から数秒の短い間隔で NIC の移譲を繰り返す必要があり，LKM を利用した移譲方式は不適当である．

LKM を利用した移譲方式ですべて初期化される NIC の制御に必要な情報は，移譲を受ける際に更新する必要があるものと，保留できるものに大別できる．パケット送受信処理時の NIC ドライバと NIC ハードウェアの処理流れを分析し，送受信パケットの参照情報と割込設定以外の情報は移譲を受ける際に保留できることが分かった．割込ルーティングの変更による移譲方式は，初期化する情報を特定する必要があり，移譲処理が複雑になるものの NIC を移譲するために必要な情報のみを更新することで，移譲処理により発生する通信停止時間を 1 ms 未満にできた．

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佐藤将也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．

参考文献

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T, H., A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, pp. 164–177 (2003).

[2] Sugerman, J., Venkitachalam, G. and Lim, B.: Virtualizing I/O Devices on VMware Workstation ’s Hosted Virtual Machine Monitor，pp. 1–14 (2001).

[3] Mattos, D. M., Ferraz, L. H. G., Costa, L. H. M. and Duarte, O. C.: Virtual network performance evaluation for future internet architectures, Vol. 4, No. 4, pp. 304–314 (2012).

[4] Fernandes, N. C., Moreira, M. D., Moraes, I. M., Ferraz, L. H. G., Couto, R. S., Carvalho, H. E., Campista E, M., Costa, H. M. K. L. and Duarte, O. C. M.: Virtual networks: Isolation, performance, and trends, Vol. 66, No. 5-6, pp. 339–355 (2011).

[5] Jones, T. M.: Linux virtualization and PCI passthrough, IBM (online), available from ⟨http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/⟩ (accessed 2016-2-2).

[6] Musleh, M., Pai, V., Walters, P, J., Younge, A. and Crago, P, S.: Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand, pp. 627–635 (2014).

[7] Jose, J., Li, M., Lu, X., Kandalla, C. K., Arnold, D. M. and Panda, D. K.: SR-IOV Support for Virtualization on Infiniband Clusters: Early experience, pp. 385–392 (2013).

[8] 千崎良太，中原大貴， 牛尾裕，片岡哲也，粟田祐一，乃村能成，谷口秀夫：マルチコアにおいて複数の Linux カーネルを走行させる Mint オペレーティングシステムの設計と評価， Vol. 110, No. 278, pp. 29–34 (2010).

[9] Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M. and Shenker, S.: Extending Networking into the Virtualization Layer (2009).

[10] Shimosawa, T. and Ishikawa, Y.: Inter-kernel communication between multiple kernels on multicore machines, Vol. 5, No. 1, pp. 13–31 (2010).

発表論文

[1] 増田陽介，乃村能成，谷口秀夫 : Mint オペレーティングシステムにおける NIC のコア間移譲方式の提案, 情報処理学会研究報告, Vol.2015-OS-132-156, No.6, pp.1-8 (2015).