Vol.2015-OS-132 No.6
2015/2/26

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

増田 陽介[1]　乃村 能成[1]　谷口 秀夫[1]

概要：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 を動作中に切り替える方式について，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 を利用した移譲方式と比較する．

1. 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 毎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，必要に応じて OS 間で I/O デバイスを移譲する手法について考察する．

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

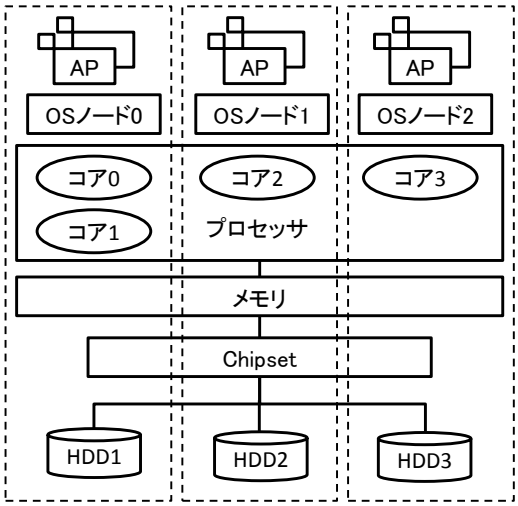

図 1　Mint の構成例

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る．このため，移譲処理に秒単位の時間がかかる．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，高速に移譲できる．

2. 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 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．

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 デバイス制御をコア間移動する．

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 OS ノードは必要に応じて 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．移譲回数が増加した場合，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，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．

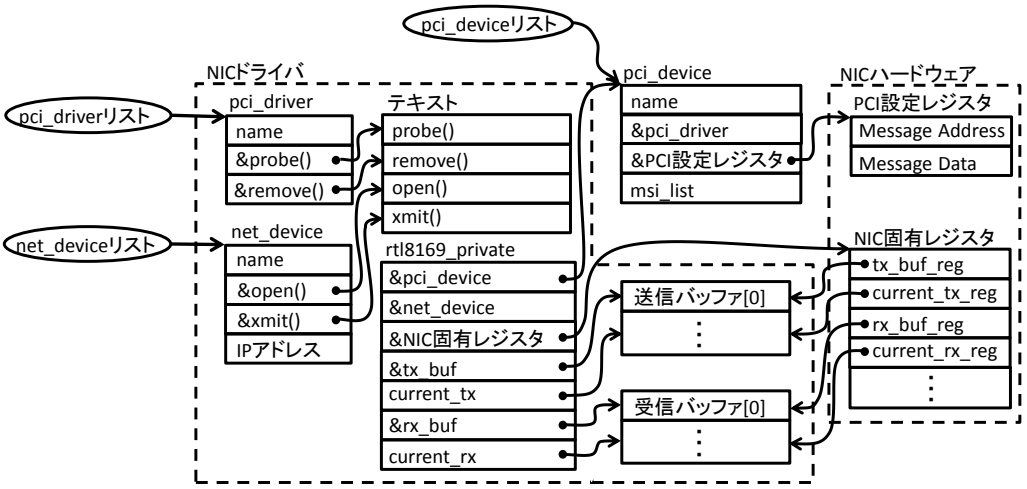

図 2　NIC の制御に必要な情報

4. LKM を利用した移譲方式

4.1 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インタフェース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．

4.2 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，4.1 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 を通信インタフェースとして利用できる．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．

4.3 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 **2** 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．

4.4 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
- (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．

(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 ドライバは送信バッファと受信バッファを作成する．
- (C) 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．
- (D) 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．
- (E) 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．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．

4.5 性能測定

4.5.1 測定対象

LKM を利用した NIC 移譲では，4.4 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．

表 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 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

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．

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．

4.5.2 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 1** に示す．同様に，パケット送信用計算機の構成を**表 2** 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．

4.5.3 測定手順

(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．

表 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 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

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．

4.5.4　測定結果

システムコールの実行時間を**表 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 4** に示す．表 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 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．

4.5.5　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** に示して，以下で説明する．

図中 (1) は表 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 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．

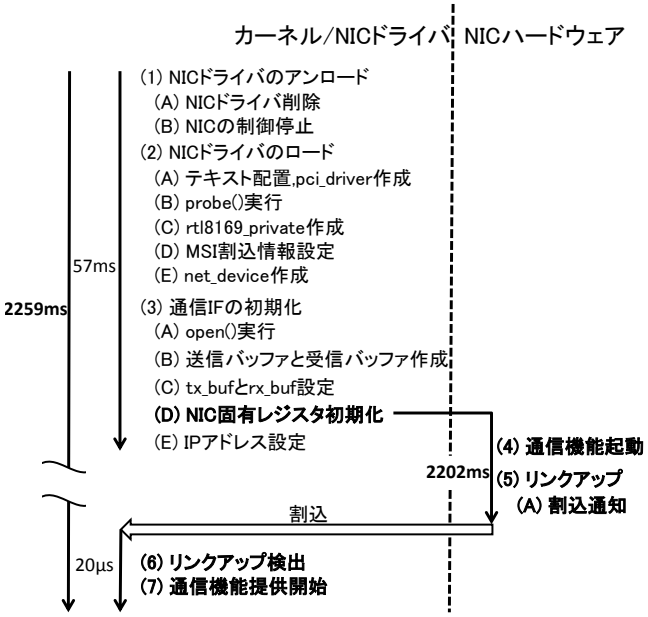

図 3　通信停止中の処理内訳

図 3 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．

5.　割込ルーティングの変更による移譲方式

5.1　更新する情報の削減による移譲処理の高速化

これまでに述べた 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．しかしながら，移譲を繰り返す状況を考えた場合，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

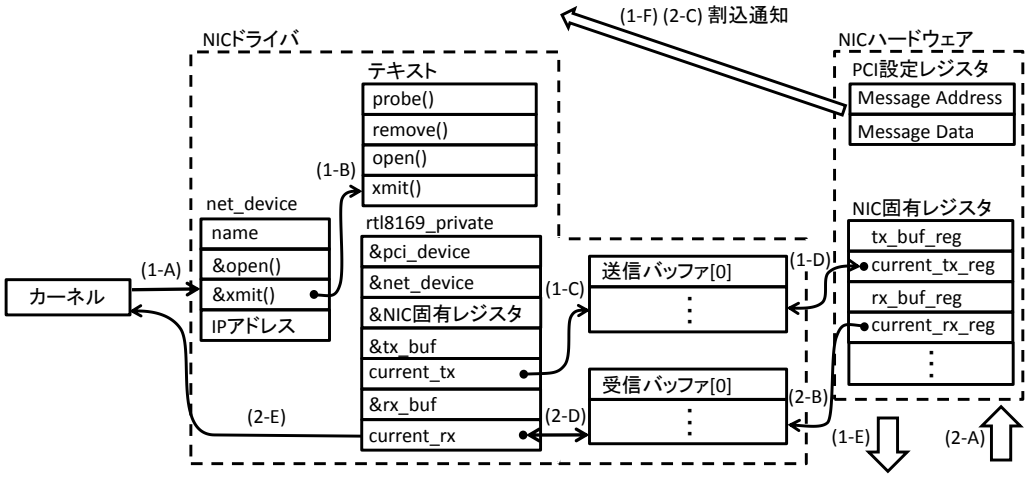

図 4　送信処理と受信処理の処理流れ

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ので，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，移譲先において NIC の状態は変化するので，再び移譲を受ける際には，その変化に追従すべく 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．あるいは，NIC ハードウェアの割込設定など，ハードウェアの設定を再設定する必要もある．

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，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 2 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，移譲を受け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，移譲を受ける際に 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 2 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 回目以降に移譲を受ける場合には，保留できる．

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，移譲を受ける際に更新が必要な情報を特定する．

5.2　移譲時に更新する必要のある情報の特定

5.2.1　送信処理と受信処理の処理流れ

図 4 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．

（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．

5.2.2 移譲時に更新が必要な情報

5.2.1 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 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．

5.3 移譲手順

割込ルーティングの変更による NIC の移譲手順を以下に示す．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 を移譲できる．

5.4 効果

割込ルーティングの変更による移譲方式は，NIC の制御に必要な情報の多くを保留したまま移譲できる．割込ルーティングの変更による移譲方式について，4.5 節と同様の方法で測定を行った．この結果，移譲処理により欠落したパケットの数が 0 であることを確認した．これにより，割込ルーティングの変更による移譲方式における通信停止時間は 1 ms 未満であることがわかる．LKM を利用した移譲方式の通信停止時間である 2,259 ms と比較して移譲時の通信停止時間を大幅に短縮できた．

6. おわりに

Mint において，NIC を利用する OS ノードを動作中に切り替える方式として，割込ルーティングの変更による移譲方式を提案し，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．

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．

今後の課題として，割込ルーティングの変更による移譲方式における NIC ハードウェアへのアクセスの排他の実現がある．

謝辞 本研究の一部は，科学研究費補助金基板研究 (B) (課題番号 24300008) による．

参考文献

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T, Ho., A., Neugebauer, R., Pratt, I., and Warfield, A.: Xen and the Art of Virtualization, Proc. the 19th ACM Symposium on Operating Systems Principles,pp.164–177, (2003).

[2] Sugerman, J.,Venkitachalam, G. and Lim, B.: Virtualizing I/O Devices on VMware Workstation’s Hosted Virtual Machine Monitor, Proc. the General Track: 2002 USENIX Annual Technical Conference, 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, Journal of Emerging Technologies in Web Intelligence, 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, Duarte, O. C. M.: Virtual networks: Isolation, performance, and trends, Annals of telecommunications-annales des tlcommunications, Vol.66, No.5–6, pp.339–355 (2011).

[5] Jones, T. M.: Linux virtualization and PCI passthrough, 入手先 (http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/) (参照 2015–01–26).

[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, Proc. Cloud Computing (CLOUD 2014) the 2014 IEEE 7th International Conference on, 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, Cluster, Cloud and Grid Computing (CCGrid) 2013 13th IEEE/ACM International Symposium on, 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, 電子情報通信学会技術研究報告, vol.110, No.278, pp.29–34 (2010).

[9] Pfaaf, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., and Shenker, S.: Extending Networking into the Virtualization Layer, Proc. 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII) (2009).

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