Mint オペレーティングシステムを用いた NIC ドライバの割り込みデバッグ手法の実現

藤田 将輝[1]　乃村 能成[1]　谷口 秀夫[1]

概要：
近年，Operating System(以下，OS) の多機能化にともなって，OS のデバッグが重要となり，活発に研究されている．特に，非同期的な処理は，常に同じタイミングで発生しないため，再現が困難である．この非同期的な処理の 1 つに割り込み処理がある．割り込み処理のデバッグを支援する方法として，仮想計算機 (以下，VM) を用いたものがある．VM を用いることの利点は 2 つある．1 つは 1 台の計算機上でデバッグを支援する機構 (以下，デバッグ支援機構) とデバッグ対象の OS(以下，デバッグ対象 OS) を動作できることである．これにより，計算機を 2 台用意するためのコストを削減できる．もう 1 つは，デバッグ支援機構をデバッグ対象 OS の外部に実装できる点である．これにより，デバッグ支援機構がデバッグ対象 OS のバグの影響を受けない．

キーワード：仮想化，割り込み，デバッグ

1. はじめに

近年，Operating System(以下，OS) の多機能化にともなって，OS のデバッグが重要となり，活発に研究されている．特に，非同期的な処理は，常に同じタイミングで発生しないため，再現が困難である．この非同期的な処理の 1 つに割り込み処理がある．割り込み処理のデバッグを支援する方法として，仮想計算機 (以下，VM) を用いたものがある．VM を用いることの利点は 2 つある．1 つは 1 台の計算機上でデバッグを支援する機構 (以下，デバッグ支援機構) とデバッグ対象の OS(以下，デバッグ対象 OS) を動作できることである．これにより，計算機を 2 台用意するためのコストを削減できる．もう 1 つは，デバッグ支援機構をデバッグ対象 OS の外部に実装できる点である．これにより，デバッグ支援機構がデバッグ対象 OS のバグの影響を受けない．デバッグ支援機構がデバッグ対象 OS へ割り込みを挿入させたり，デバッグ支援機構がデバッグ対象 OS の動作を再現したりすることで，バグを再現し，デバッグを支援する．しかし，VM を用いる場合，VM とハイパーバイザ間の処理の遷移に伴う処理負荷が存在する．このため，VM を用いたデバッグ支援環境では一定間隔で発生する割り込みや，短い間隔で発生するバグのように，処理負荷が影響する割り込み処理のデバッグが困難である．

そこで，Multiple Independent operating systems with New Technology(以下，Mint)[1] を用いたデバッグ手法が提案されている．Mint は仮想化を用いずに複数の Linux を動作できる．Mint を用いてデバッグ支援環境を構築すると，ハイパーバイザが存在しないため，処理の遷移に伴う処理負荷も存在しなくなる．これにより，一定間隔で発生する割り込みや短い間隔で発生するバグのデバッグが可能になる．

本研究では，提案手法を用いて非同期的な割り込みが頻繁に発生する NIC ドライバを対象としたデバッグ支援環境の実装について述べる．これにより，Mint における割り込み処理のデバッグ支援環境で，割り込み処理が再現できることを示す．

2. 関連研究

2.1 VM を用いたデバッグ支援機構

OS の割り込みのデバッグを支援する環境の既存研究として VM を用いたものがある．VM を用いた割り込みデバッグ支援環境は大きく分けて 2 つある．割り込み挿入法 [2] とロギング/リプレイ手法 [3][4][5] である．これらの概要について以下で説明する．

割り込み挿入法

割り込み挿入法はデバッグ対象 OS の他に，デバッグ

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

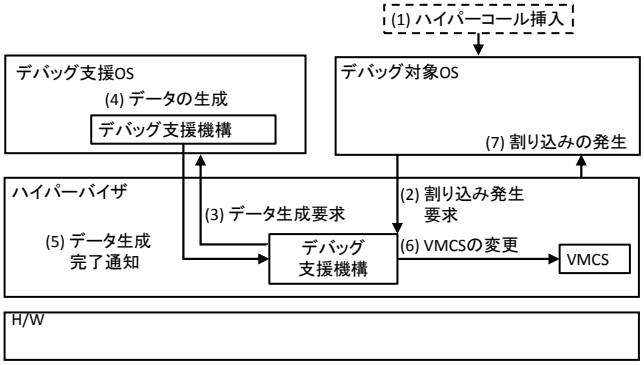

図 1 割り込み挿入法の処理流れ

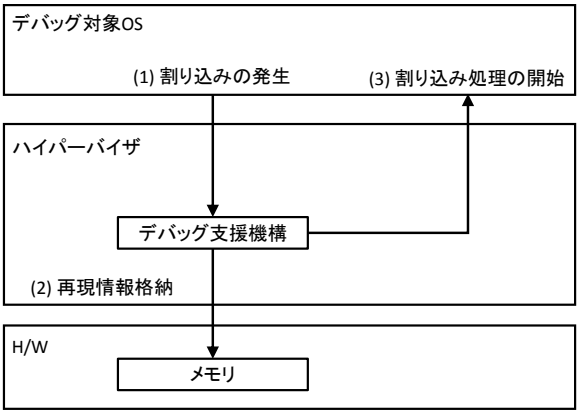

図 2 ロギングの処理流れ

支援機構として，ハイパーバイザとデバッグを支援する OS(以下，デバッグ支援 OS) が走行する．ユーザがデバッグ対象 OS の割り込みを挿入したいコード位置にハイパーコールを挿入する．デバッグ対象 OS の走行時に，ハイパーコールを挿入した位置で処理がハイパーバイザに遷移する．そして，デバッグ支援 OS で割り込みに必要なデータを用意した後，デバッグ対象 OS に割り込みを発生させ，バグを再現する．これにより，デバッグを支援する．割り込み挿入法を用いた既存研究として仮想マシンモニタを用いた割り込み処理のデバッグ手法 [2] がある．これは仮想マシンモニタがデバッグ対象 OS に仮想的な割り込みを発生させるものである．

ロギング/リプレイ手法

ロギング/リプレイ手法はデバッグ対象 OS の他に，デバッグ支援機構として，ハイパーバイザが走行する．この手法はデバッグ対象 OS がバグを起こすまでの流れを保存 (ロギング) し，再現 (リプレイ) することで，デバッグを支援する．また，動作の流れを再現するための情報 (以下，再現情報) として，以下の 2 つがある．

(1) 割り込みの種類，割り込み発生アドレス，および分岐命令を経由した回数
(2) キーコードや，パケットなどのような割り込み処理で扱うデータ

ロギング/リプレイ手法を用いた既存研究として TTVM[3]，Aftersight[4] および Sesta[5] がある．TTVM は再現情報に加え，デバッグ対象 OS 側の VM の状態を保存する．Aftersight はロギングとリプレイを異なる種類のハイパーバイザで行う．Sesta はロギングを行う OS の処理を追うようにしてリプレイを行う OS を走行させる．

2.2 割り込み挿入法の処理流れ

割り込み挿入法における割り込みは，ユーザが割り込みを発生させたいコード位置にハイパーコールを挿入することで発生させる．割り込みは挿入したコード位置で発生する．この際，割り込みは Virtual Machine Control Structure(以下，VMCS) と呼ばれるデータ構造の値を書き換えることで発生させられる．デバッグ対象 OS とデバッグ支援 OS はハイパーバイザ内のデバッグ支援機構でデータの授受をする．割り込み挿入法の処理流れについて図 1 に示し，以下で説明する．

(1) ユーザが割り込みを挿入したい位置にハイパーコールを挿入する．この際，割り込みの種類とデータを指定する．
(2) デバッグ対象 OS に挿入したハイパーコールにより，デバッグ対象 OS がハイパーバイザのデバッグ支援機構へ割り込み発生要求を行う．その後，デバッグ対象 OS の処理を中断し，ハイパーバイザへ処理が遷移する．
(3) ハイパーバイザのデバッグ支援機構がデバッグ支援 OS のデバッグ支援機構へ割り込みに必要なデータの生成要求を行う．
(4) デバッグ支援 OS のデバッグ支援機構が割り込みに必要なデータを生成する．
(5) デバッグ支援 OS のデバッグ支援機構がハイパーバイザのデバッグ支援機構へデータの生成完了を通知する．
(6) ハイパーバイザのデバッグ支援機構が VMCS の値を変更する．これにより，処理がハイパーバイザからデバッグ対象 OS へ処理が遷移するとき割り込みが発生する．
(7) デバッグ対象 OS へ処理が遷移し，割り込みが発生する．

このように，ハイパーバイザへの処理の遷移のため，処理負荷が発生しており，短い間隔や一定間隔の割り込み挿入が困難である．

2.3 ロギング/リプレイ手法の処理流れ

2.3.1 ロギングの処理流れ

ロギングの処理流れについて図 2 に示し，以下で説明する．

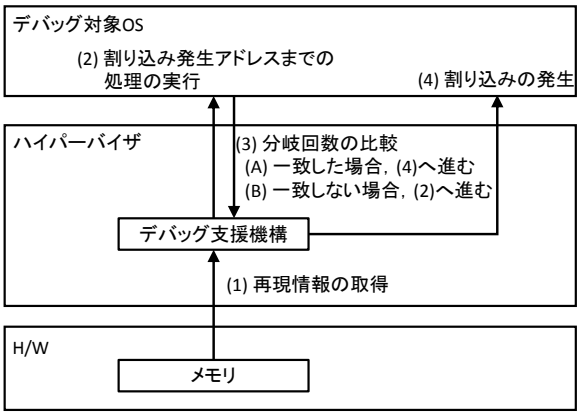

図 3 リプレイの処理流れ

(1) デバッグ対象 OS に割り込みが発生すると，処理を中断し，ハイパーバイザに処理が遷移する．
(2) ハイパーバイザのデバッグ支援機構が再現情報をメモリに格納する．
(3) ハイパーバイザからデバッグ対象 OS へ処理が遷移し，デバッグ対象 OS が中断していた割り込み処理を再開する．

2.3.2 リプレイの処理流れ

リプレイの処理流れについて図 3 に示し，以下で説明する．

(1) ハイパーバイザのデバッグ支援機構がメモリから再現情報を取得する．
(2) 取得した再現情報よりデバッグ対象 OS が割り込みが発生するアドレスまで命令を実行する．
(3) ハイパーバイザのデバッグ支援機構が再現情報の分岐回数と，現在のデバッグ対象 OS の分岐回数を比較する．比較結果により，以下の処理に分岐する．
  (a) 一致した場合，(4) へ進む．
  (b) 一致しない場合，(2) へ進む．
(4) デバッグ対象 OS へ割り込みが発生する．

このように，割り込みの発生タイミングを調整しようとした場合，ユーザが再現情報を指定する必要があり，任意のタイミングの割り込み発生が困難である．また，ロギング時にハイパーバイザへ処理が遷移するため，この負荷により，ロギング中に実計算機上で発生する間隔での割り込みは発生しないと考えられる．したがって，実計算機上で発生する間隔での割り込みの再現情報が得られず，再現が困難である．

2.4 問題点

割り込み挿入法の問題点について以下で説明する．

(問題点 1) 実計算機上で発生する間隔での複数割り込みの発生が困難

割り込み挿入法では割り込みを発生させる際，OS のコードの任意の位置にハイパーコールを挿入することで割り込みを発生させる．ハイパーコールが実行されるタイミングは OS の処理速度に依存する．このため，複数の割り込みを CPU へ発生させる間隔の調整は，ハイパーコールの間隔を調整することで行う．ユーザがハイパーコールの間隔を調整することで CPU へ発生する間隔を調整することは非常に困難である．つまり，実計算機上で発生する間隔での複数の割り込み (以下，実割り込み) を発生させることが困難である．

また，ロギング/リプレイ手法の問題点について以下で説明する．

(問題点 2) 任意のタイミングでの割り込み発生が困難

ロギング/リプレイ手法は，ロギング時に発生した割り込みに対する処理をリプレイ時に確認できる．しかし，任意のタイミングで割り込みを発生させるためには，再現情報として割り込みを発生させるアドレスと分岐回数をユーザが用意しなければならない．これらの指定が困難であるため，任意のタイミングで割り込みを発生させることが困難である．

(問題点 3) 実割り込みの発生が困難

ロギング/リプレイ手法は，ロギングにおけるデバッグ対象 OS とハイパーバイザ間の処理の遷移や再現情報の格納による処理負荷が発生する．このため，実割り込みがロギング中に発生しないと考えられる．ロギング中に実割り込みが発生しない場合，実割り込みを再現するための再現情報を保存できない．このため，実割り込みの発生が困難である．

これらの問題点から，割り込み処理のデバッグには，デバッグ対象 OS がデバッグ支援機構の処理負荷の影響を受けない環境と任意のタイミングで割り込みを発生できる環境が必要である．また，実際の割り込み処理の再現をするため，これらの環境は実際の割り込み処理と同様の挙動をする必要がある．

3. Mint オペレーティングシステム

3.1 Mint の設計方針

Mint とは 1 台の計算機上で仮想化を用いずに計算機資源を論理分割することによって複数の Linux を動作させる方式である．Mint の設計方針として以下の 2 つが挙げられる．

(1) 全ての Linux が相互に処理負荷の影響を抑制
(2) 全ての Linux が入出力性能を十分に利用可能

3.2 Mint の構成

Mint では，1 台の計算機上でプロセッサ，メモリ，およびデバイスを分割し，各 OS が占有する．Mint の構成例を図 4 に示し，説明する．本稿では Mint を構成する OS を OS ノードと呼ぶ．Mint では，最初に起動する OS を OS ノード 0 とし，起動順に OS ノード 1，OS ノード 2,... と

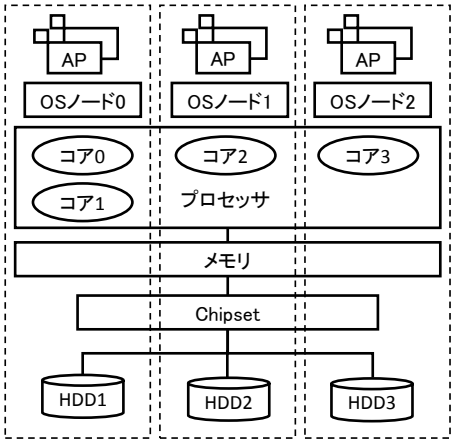

図 4 Mint の構成

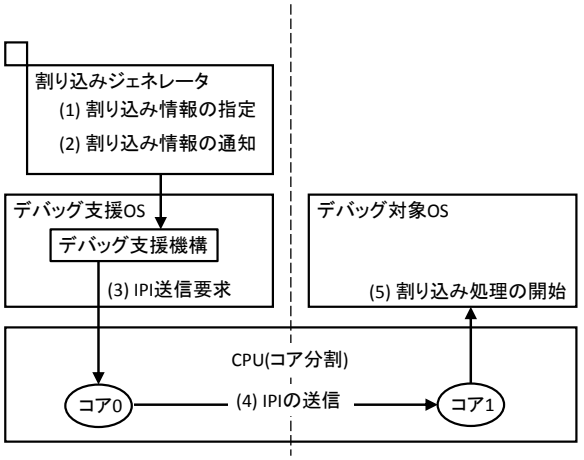

図 5 Mint を用いたデバッグ支援環境の処理流れ

する．

(1) プロセッサ
コア単位で分割し，各 OS ノードがコアを 1 つ以上占有する．

(2) メモリ
空間分割し，各 OS ノードが分割領域を占有する．

(3) デバイス
デバイス単位で分割し，各 OS ノードが指定されたデバイスを占有する．

3.3 Mint を用いたデバッグ支援環境

3.3.1 方針

2.4 節に示した VM を用いた割り込みデバッグ手法の問題点を解決する手段としての Mint を用いたデバッグ支援環境の方針について以下に示し，説明する．

(方針 1) 実割り込みを発生させる環境の提供
割り込みの発生間隔に依存するバグを確認するには，実割り込みを発生させる必要がある．しかし，VM を用いた既存研究では 2.4 節で述べた問題により，実割り込みの発生が困難である．そこで Mint を用いた割り込み処理のデバッグ支援環境では，デバッグ対象 OS へ実割り込みを発生させる環境を提供する．

(方針 2) 任意のタイミングで割り込みを発生させる環境の提供
デバッグの際，デバッグ対象処理のバグの有無やバグの発生箇所を確認するために，デバッグ対象の処理を繰り返し実行する．しかし，割り込み処理は非同期な処理であるため繰り返し実行することが困難である．そこで Mint を用いた割り込み処理のデバッグ支援環境では，任意のタイミングで割り込みを発生できる環境を提供する．

3.3.2 構成と処理流れ

Mint を用いたデバッグ支援環境について図 5 に示し，説明する．Mint 上でデバッグ支援 OS を OS ノード 0，デバッグ対象 OS を OS ノード 1 として動作させる．デバッグ支援 OS はコア 0 を占有し，割り込みジェネレータとデバッグ支援機構を持つ．デバッグ対象 OS はコア 1 を占有する．この構成で下記の処理を行うことにより，実割り込みを発生させる．

(1) デバッグ支援 OS 上で動作するアプリケーション (以下，AP) である割り込みジェネレータを用いてユーザが割り込み情報を指定する．

(2) 割り込みジェネレータがシステムコールを用いてデバッグ支援機構を呼び出す際，指定した割り込み情報をデバッグ支援 OS のデバッグ支援環境に通知する．

(3) デバッグ支援機構がコア 0 へ Inter-Processor Interrupt(以下，IPI) の送信要求を行う．

(4) コア 0 が IPI の送信要求を受けると，コア 1 へ IPI を送信する．

(5) コア 1 が IPI を受信すると割り込み処理が開始する．

3.4 Linux 改変による割り込み処理の挙動への影響

Mint では 1 台の計算機上で複数の Linux を動作させるため，各 Linux に改変を加えている [6]．この際の改変は各 Linux の起動時に認識するプロセッサ，メモリ，およびデバイスを調停するためのものである．起動終了処理のみ変更を加えており，それ以外の機能は改変前の Linux と同様に動作する．したがって，Mint における割り込み処理は，改変前の Linux の割り込み処理と同等であるといえる．

4. NIC ドライバの割り込みデバッグ環境の設計

4.1 目的

割り込み処理の 1 つに，デバイスドライバの割り込み処理がある．デバイスドライバの割り込み処理は，デバイスが OS へ非同期的に発生させる割り込みにより実行される．NIC では頻繁に通信を行なっているため，割り込み処理も頻繁に行われている．したがって，実割り込みも頻繁に発

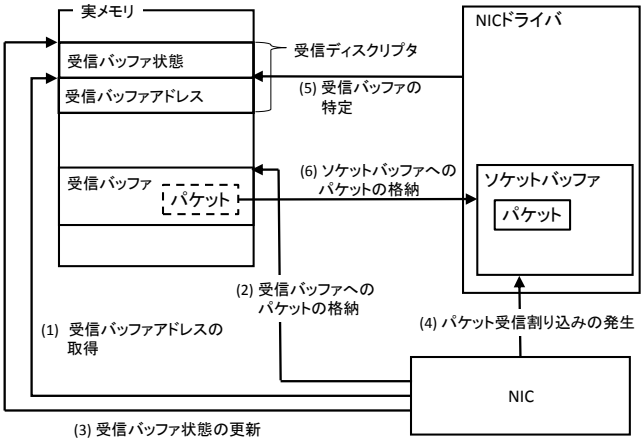

図 6 NIC ドライバのパケット受信処理流れ

生していると考えられる．そこで，NIC ドライバを対象とした Mint を用いた割り込みデバッグ支援環境を構築し，割り込み処理を再現できる環境を実現する．これにより，Mint における割り込み処理のデバッグ支援環境で，割り込み処理を再現する．

4.2 NIC ドライバのパケット受信の流れ

本研究では例として，NIC がパケットを受信した際に発生させる割り込み (以下，パケット受信割り込み) に対する NIC ドライバの割り込み処理のデバッグ支援環境を実装する．NIC によるパケットの格納と割り込み処理について図 6 に示し，以下で説明する．受信バッファは NIC が受信したパケットを格納するメモリ領域である．また，受信ディスクリプタは受信バッファへのアドレスと受信バッファがパケットを受信済みか否かの状態 (以下，受信バッファ状態) を保持するメモリ領域である．

(1) NIC が受信ディスクリプタから受信バッファのアドレスを取得する．これにより，NIC が受信バッファにパケットを格納可能になる．
(2) NIC が受信バッファのアドレスをもとに，受信バッファへパケットを格納する．
(3) NIC が受信ディスクリプタ中の受信バッファ状態を更新し，受信済み状態にする．これにより，NIC ドライバが受信バッファがパケットを受信済みか否かを判別可能になる．
(4) NIC が NIC ドライバに割り込みを発生させる．これにより，OS が割り込み処理を開始する．
(5) NIC ドライバが受信ディスクリプタの受信バッファ状態を確認する．これにより，受信バッファを特定する．
(6) NIC ドライバが受信バッファからパケットを取得し，ソケットバッファへ格納する．

4.3 設計方針

Mint を用いた NIC ドライバの割り込みデバッグ支援環境の設計方針について以下に示し，説明する．

(方針 1) 実割り込みの再現
デバッグ支援 OS がデバッグ対象 OS の NIC ドライバへ実割り込みを発生させられる環境を構築する．従来のデバッグ支援環境では，(問題 1) と (問題 3) がある．そこで NIC ドライバの実割り込みを再現する．

(方針 2) NIC の動作をデバッグ支援 OS が再現
NIC ドライバの割り込み処理のデバッグを対象とするため，ハードウェア (NIC) のバグは考慮しない．したがって，NIC を用いずに割り込みデバッグ支援環境を構築する．NIC はパケットを受信バッファに格納し，受信ディスクリプタ中の受信バッファ状態を更新する．NIC を用いずに割り込みデバッグ支援環境を構築するため，デバッグ支援 OS により NIC を再現する．これにより，NIC を用いずに NIC ドライバの割り込みデバッグ支援環境を構築する．

(方針 3) 共有メモリを用いたパケットの受け渡し
NIC のパケット受信割り込みを再現するため，デバッグ対象 OS でパケットを受け取る必要がある．デバッグ対象 OS がパケットを受け取るには，入出力デバイスかメモリを用いる．しかし．本研究ではデバイスのバグを考慮しないため，入出力デバイスは用いない．そこで，デバッグ支援 OS とデバッグ対象 OS 間の共有メモリを用いる．共有メモリ上に NIC の受信バッファと受信ディスクリプタを配置することにより，デバッグ支援 OS からデバッグ対象 OS の NIC ドライバへパケットを受け渡す．

4.4 課題

それぞれの設計に対する課題について以下に示し，説明する．また，各課題の項目の末尾にどの方針に対する課題かを示す．

(課題 1) 割り込み間隔，回数の調整 (方針 1)
実割り込みを再現するため，割り込みの間隔を調整し，連続で割り込みを発生させる必要がある．また，割り込みの間隔と回数を指定できる必要がある．

(課題 2) パケットの作成 (方針 2)
NIC ドライバのパケット受信処理を再現するため，処理させるパケットを作成する必要がある．

(課題 3) パケットの格納 (方針 2)
NIC のパケットを受信バッファに格納する機能を再現するため，デバッグ支援 OS が受信バッファにパケットを格納する必要がある．

(課題 4) 受信バッファ状態の更新 (方針 2)
NIC の受信ディスクリプタ中の受信バッファ状態を受信済み状態に書き換える機能を再現するため，デバッグ支援 OS が受信バッファ状態を書き換える必要がある．

(課題 5) 割り込みの発生 (方針 2)
NIC の NIC ドライバに割り込みを発生させる機能を再現するために，NIC 以外のものから割り込みを発生させる必要がある．

(課題 6) 割り込みハンドラの作成 (方針 2)
NIC からのパケット受信割り込みでないため，NIC ドライバ本来の割り込みハンドラは動作しない．したがって，割り込みの契機を変更する必要がある．また，変更した割り込み契機により動作する割り込みハンドラを用意する必要がある．

(課題 7) 受信バッファの作成 (方針 3)
デバッグ支援 OS が共有メモリにパケットを配置し，デバッグ対象 OS が共有メモリからパケットを取得するため，共有メモリに NIC の受信バッファを作成する必要がある．

4.5 対処

課題への対処を以下に示し，説明する．また，各対処の項目の末尾に，どの課題に対する対処かを示す．

(対処 1) 割り込みジェネレータによる指定 (課題 1)
割り込み間隔と回数をユーザが指定できるようにするため，デバッグ支援 OS 上にこれらの情報が指定できる割り込みジェネレータを AP として実装する．指定した間隔と回数で割り込みを発生させるシステムコールを発行する．

(対処 2) システムコール内でのパケットの作成 (課題 2)
NIC ドライバが処理するパケットを作成する．具体的にはパケットの種類に応じたヘッダをデータに付与する．

(対処 3) システムコールによるパケットの格納 (課題 3)
デバッグ支援 OS のシステムコールにより，NIC の受信バッファへ (対処 2) のパケットを格納する．

(対処 4) システムコールによる受信バッファ状態の変更 (課題 4)
受信バッファ状態を書き換えるため，受信ディスクリプタを共有メモリに配置し，デバッグ支援 OS とデバッグ対象 OS の両 OS で参照可能にする．これにより，デバッグ支援 OS が受信ディスクリプタ中の受信バッファ状態を書き換え可能にする．

(対処 5) 割り込み契機として IPI を使用 (課題 5)
NIC の受信割り込みの再現として，コア間割り込みである IPI を使用する．これにより，(問題 1) と (問題 3) を解決できる．

(対処 6) IPI を契機とした割り込みハンドラ (課題 6)
割り込みの契機を IPI に変更したことにより，IPI により動作する NIC ドライバの割り込みハンドラを作成する必要がある．この割り込みハンドラはデバッグ対象 OS が占有するコアが IPI を受信すると動作し，

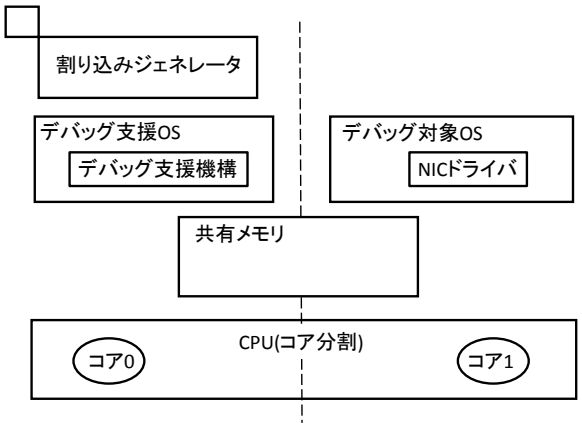

図 7 NIC ドライバのデバッグ支援環境の概要

共有メモリの受信バッファからパケットを取得し，ソケットバッファに格納する機能を持つ．この割り込みハンドラを NIC ドライバの初期化処理の中で登録し，使用可能にする．

(対処 7) 共有メモリへ受信バッファの作成 (課題 7)
共有メモリを用いてデバッグ支援 OS とデバッグ対象 OS の NIC ドライバ間でパケットを受け渡すため，NIC の受信バッファを共有メモリに作成する必要がある．このため，NIC ドライバの初期化処理内で，受信バッファのアドレスを変更し，共有メモリのアドレスにする．

4.6 NIC ドライバのデバッグ支援環境の構成と機能

Mint 上でデバッグ支援 OS を OS ノード 0，デバッグ対象 OS を OS ノード 1 として動作させる．デバッグ支援 OS はコア 0 を占有し，割り込みジェネレータとデバッグ支援機構を持つ．デバッグ対象 OS はコア 1 を占有し，NIC ドライバを持つ．また，メモリ上には両 OS がアクセスできる共有メモリ空間が存在する．割り込みジェネレータ，デバッグ支援機構，および NIC ドライバの構成と機能について図 7 に示し，以下で説明する．また，各機能の項目の末尾にどの対処に対する機能かを示す．

(1) 割り込みジェネレータ
割り込みジェネレータはデバッグ支援 OS 上で動作する AP である．割り込みジェネレータは以下の機能を持つ．

(機能 1) 割り込み情報の指定と通知 (対処 1)
デバッグ支援 OS 上で動作する AP である．この AP で割り込みの間隔，割り込みの回数を指定する機能を実装する．システムコールにより，デバッグ支援機構を呼び出す際にこの情報をデバッグ支援機構に通知する．

(2) デバッグ支援機構
デバッグ支援機構はデバッグ支援 OS が持つ機構である．これはシステムコールにより実現する．デバッグ

支援機構は以下の機能を持つ．

(機能 2) パケットの作成
通知された割り込み情報により，パケットを作成する．パケットとしてデータを定義し，これにヘッダを付与することでパケットを作成する．

(機能 3) 受信バッファへのパケットの格納 (対処 3)
機能 2 で作成したパケットを共有メモリに作成された受信バッファに格納する．

(機能 4) 受信バッファ状態の更新 (対処 4)
共有メモリに配置されている受信ディスクリプタを取得し，その受信ディスクリプタ中の受信バッファ状態を書き換え，受信済み状態にする．

(機能 5) IPI の送信 (対処 5)
デバッグ支援 OS が占有しているコア 0 にコア 0 からデバッグ対象 OS が占有しているコア 1 へ IPI を送信する要求を発行する．

(3) NIC ドライバ
デバッグ対象 OS 内で動作している．また，本環境には NIC を用いないため，以下の機能を持つように改変されている．

(機能 6) 割り込みハンドラ (対処 6)
デバッグ対象 OS で動作する NIC ドライバにおいて IPI により動作し，共有メモリに作成された受信バッファからパケットを取得し，ソケットバッファに格納する割り込みハンドラである．NIC ドライバの初期化処理でこの割り込みハンドラを登録する．

(機能 7) 受信バッファの作成 (対処 7)
デバッグ対象 OS で動作する NIC ドライバの初期化処理中で受信バッファのアドレスを決定する際に，このアドレスを共有メモリのアドレスに変更する．

5. 実装

5.1 NIC ドライバのデバッグ支援環境の処理流れ

Mint を用いた NIC ドライバの割り込みデバッグ支援環境の処理流れを図 8 に示し，以下で説明する．

(1) ユーザが割り込みジェネレータを用いて割り込み情報を指定する．
(2) 割り込みジェネレータがデバッグ支援 OS のデバッグ支援機構をシステムコールを用いて呼び出す．この際 (1) で指定した割り込み情報を共に通知する．
(3) デバッグ支援機構が割り込み情報からパケットを作成する．
(4) デバッグ支援機構が共有メモリの受信バッファへパケットを格納する．
(5) デバッグ支援機構が共有メモリの受信ディスクリプタの受信バッファ状態を更新する．
(6) デバッグ支援機構がコア 0 へ IPI 送信要求を行う．
(7) コア 0 がコア 1 へ IPI を送信する．

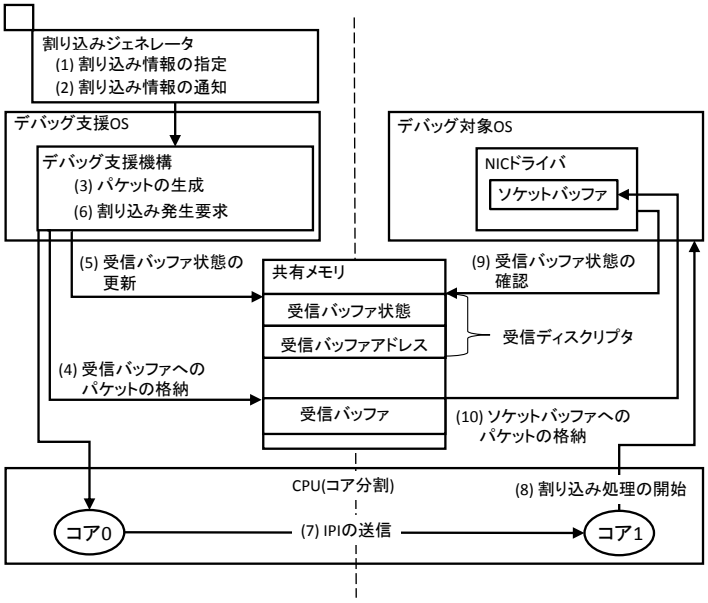

図 8 NIC ドライバのデバッグ支援環境の処理流れ

(8) コア 1 が IPI を受信すると，デバッグ対象 OS の割り込みハンドラが動作する．
(9) NIC ドライバが共有メモリの受信ディスクリプタ中の受信バッファ状態を確認する．
(10) NIC ドライバが共有メモリの受信バッファからパケットを取得し，ソケットバッファに格納する．

5.2 必要な機能の実現

割り込み情報の通知

割り込みジェネレータをデバッグ支援 OS 上で動作する AP として実装する．割り込みジェネレータでは割り込みを発生させる際の情報を指定する．具体的には，割り込みの種類，割り込みの間隔，および回数である．システムコールを用いてデバッグ支援機構を呼び出す際に，これらの情報を共に通知する．

パケットの作成

デバッグ支援機構が NIC ドライバに割り込み処理をさせるためにパケットを作成する．パケットの種類は，UDP としている．パケットの種類に応じたヘッダをデータに付与することでパケットを作成する．

受信バッファへのパケットの格納

Mint の共有メモリを利用してパケットの受け渡しを実現する．デバッグ支援機構において，作成されたパケットを共有メモリの受信バッファに格納する．共有メモリへのパケットの格納はシステムコールとして定義されたデバッグ支援機構によって実現する．

受信バッファ状態の更新

受信バッファに関する情報は受信ディスクリプタが保持している．具体的には受信バッファのアドレス，および受信バッファ状態を保持している．NIC ドライバはパケット

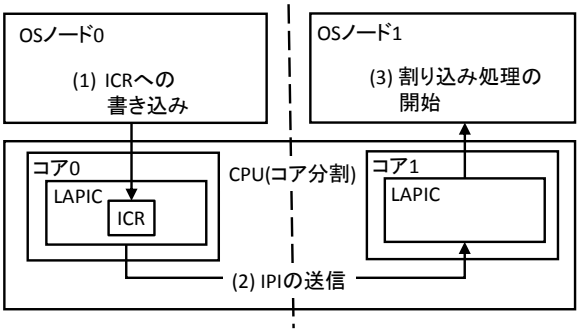

図 9 IPI の送信

の受信処理を行う際，以下の流れでパケットを取得する．

(1) NIC ドライバが受信ディスクリプタ中の受信バッファ状態を確認する．
(2) 受信バッファ状態が受信済み状態ならば NIC ドライバは受信ディスクリプタから受信バッファのアドレスを取得する．
(3) NIC ドライバが取得した受信バッファのアドレスから受信バッファにアクセスし，パケットを取得する．

本来は NIC がパケットを受信バッファに格納した際にこの受信バッファ状態を変更する．本環境では NIC を使用せず，デバッグ支援 OS が NIC の動作を再現する．このため，デバッグ支援 OS のデバッグ支援機構が共有メモリにパケットを格納した際にこの受信バッファ状態を更新する．また，デバッグ支援 OS が受信ディスクリプタを参照可能にするため，共有メモリに受信ディスクリプタを配置する必要がある．そこで，NIC ドライバの初期化処理中の受信ディスクリプタのアドレスを決定する処理中で共有メモリのアドレスに変更する．これにより，共有メモリに受信ディスクリプタが作成される．

IPI の送信

NIC を用いずにデバッグ対象 OS に割り込みを発生させるため，IPI を用いる．デバッグ支援 OS でシステムコールとして定義されているデバッグ支援機構がデバッグ支援 OS が占有しているコアに IPI の送信要求を発行する．これをコアが受け取ることで IPI が送信される．デバッグ対象 OS で登録した割り込みハンドラのベクタ番号を IPI で指定することで登録した割り込みハンドラが動作する．IPI を送信する際の流れを図 9 に示し，以下で説明する．

(1) コアの持つ LAPIC(割り込みコントローラ) 中の ICR という IPI 送信用のレジスタに，ベクタ番号と LAPIC ID を書き込む．
(2) LAPIC ID を参照し，この ID を持つコアへ IPI を送信する．
(3) コアが IPI を受信すると，指定したベクタ番号に対応した割り込みハンドラが動作する．

共有メモリからパケットを取得する割り込みハンドラ

割り込みの契機を変更したことにより，NIC ドライバの割り込みハンドラを変更する必要がある．このため，IPI により動作し，共有メモリの受信バッファからパケットを取得し，NIC ドライバのソケットバッファに格納する割り込みハンドラを作成する．IPI を受信してから NIC ドライバのソケットバッファにパケットを格納するまでの流れを以下に示し，説明する．

(1) デバッグ対象 OS が占有するコア 1 が IPI を受信する．
(2) 割り込みハンドラが動作し，共有メモリの受信ディスクリプタ内の受信バッファ状態を確認する．
(3) 受信バッファ状態が受信済みの状態であれば，受信済みである受信バッファのアドレスの受信バッファからパケットを取得し，ソケットバッファに格納する．

作成した割り込みハンドラを利用するには，割り込みハンドラを登録する必要がある．したがって NIC ドライバの初期化処理中で作成したハンドラを登録する．この際，NIC ドライバのプライベート構造体を参照できるようにするため，NIC ドライバの初期化処理の関数内で NIC のデバイス構造体を指定して登録する．

受信バッファの作成

Mint の共有メモリを用いてデバッグ支援 OS からデバッグ対象 OS の NIC ドライバへパケットを受け渡すために，共有メモリに NIC の受信バッファを作成する必要がある．これを実現するため，NIC ドライバの初期化処理中の受信バッファのアドレスを決定する処理内で共有メモリのアドレスに変更する．これにより，共有メモリに受信バッファが作成される．

6. 評価

6.1 評価目的

本評価の目的は，Mint を用いたデバッグ支援環境を使用して，デバッグ対象 OS にパケット受信割り込み処理をさせた際に，どの程度の割り込み間隔とパケットのサイズならば正常にパケットを処理できるかを測定し，実現できるスループットを算出することにより，実際の NIC を再現することに十分な性能を実現できているかを評価することである．

6.2 評価環境

本評価における評価環境を表 1 に示す．

表 1 測定環境

項目名	環境
OS	Fedora14 x86_64(Mint 3.0.8)
CPU	Intel(R) Core(TM) Core i7-870 @ 2.93GHz
メモリ	2GB
Chipset	Intel(R) 5 Series/3400
NIC ドライバ	RTL8169
ソースコード	r8169.c

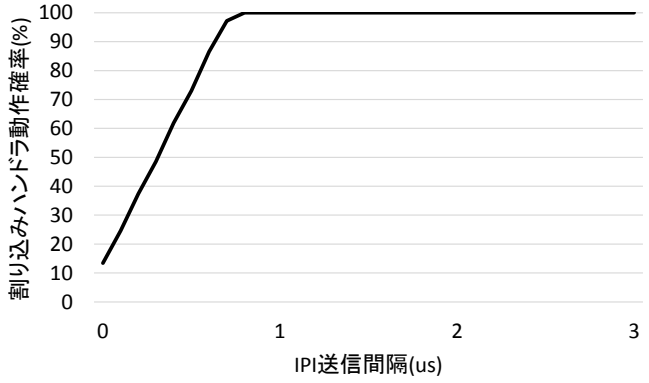

図 10 IPI の反応限界

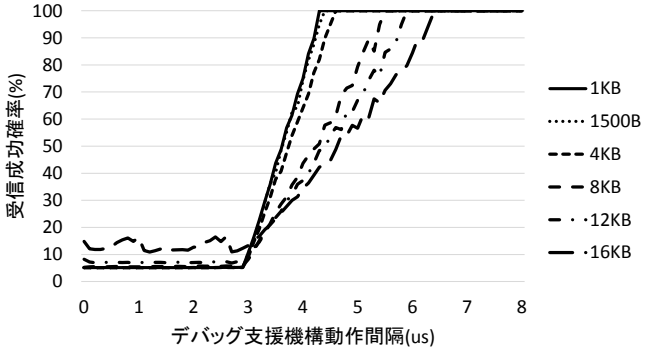

図 11 NIC ドライバにおける受信成功確率

6.3 測定対象

本評価で行う測定は，以下の 3 つである．

【測定 1】 コアが反応可能な IPI の受信間隔の測定

【測定 2】 NIC ドライバがパケットを処理可能なデバッグ支援機構の動作間隔とパケットサイズの測定

【測定 3】 デバッグ対象 OS 上で動作する UDP の受信プログラムがパケットを処理可能なデバッグ支援機構の動作間隔とパケットサイズの測定

6.4 測定 1

6.4.1 測定方法

IPI の間隔を調整し，連続で IPI を送信し，割り込みハンドラの動作確率を測定する．具体的には，デバッグ支援 OS からデバッグ対象 OS に 5000 回連続で IPI を送信し，デバッグ対象 OS の割り込みハンドラを動作させた際に，カウンタをインクリメントすることで，動作した割り込みハンドラの回数を測定し，動作確率を算出する．これらの動作を 1 サイクルとし，IPI の送信間隔を一定時間増加させながら何サイクルも行う．これにより，全ての IPI にコアが反応できる送信間隔を調査する．

6.4.2 測定結果

測定結果を図 10 に示し，分かったことを以下で説明する．

(1) IPI の送信間隔が 0us から 1us の間，割り込みハンドラの動作確率は一次関数的に増加する．

(2) IPI の送信間隔を 1us 以上にすると動作確率は 100%となる．

6.5 測定 2

6.5.1 測定方法

本デバッグ支援環境を用いて，連続でパケットを送信した際の NIC ドライバでパケットを処理できる確率を測定する．測定に用いたパケットは UDP パケットの Ethernet フレームであり，パケットのサイズは Ethernet フレームのサイズである．具体的には以下の方法で NIC ドライバでのパケットの受信成功確率を測定する．

(1) デバッグ支援機構の動作間隔を指定し，デバッグ支援機構を 5000 回動作させる．

(2) NIC ドライバでパケットを処理する際，共有メモリに配置したカウンタをインクリメントする．

(3) デバッグ支援機構を 5000 回動作させた後，デバッグ支援 OS が共有メモリに配置したカウンタを取得し，成功回数を求める．この回数とデバッグ支援機構動作回数である 5000 回から受信成功率を算出する．

この流れを 1 サイクルとし，1 サイクル毎にデバッグ支援機構の動作間隔を 100ns 増加させながら何サイクルも行う．これにより，各サイクルのデバッグ支援機構の動作間隔でのパケット受信成功確率を求められる．また，これらの操作を以下の 6 つのサイズのパケットで行うことで，各パケットのサイズにおける受信可能な動作間隔がわかる．

(1) 1KB

(2) 1500B

(3) 4KB

(4) 8KB

(5) 12KB

(6) 16KB

さらに，1 サイクルにかかった時間を測定する．この時間と，処理したデータ量からどの程度の通信量を実現できているかが分かる．

6.5.2 測定結果

測定結果を図 11 に示し，読み取れることを以下に示す．

(1) 0us から 3us の間，どのサイズでもほとんどの割り込みでパケット受信に失敗している．

(2) 3us を超えると，一次関数的に受信成功確率は増加し始める．この際，パケットサイズが小さいほど傾きが急になっている．

(3) 受信成功確率が 100%になってからはどれだけ動作間隔を増加させても受信成功確率は 100%となる．

6.6 測定 3

6.6.1 測定方法

本デバッグ支援環境を用いて，デバッグ支援 OS からデバッグ対象 OS へパケットを送信した際の，デバッグ対象 OS 上で動作する UDP の受信用プログラムでどの程度の

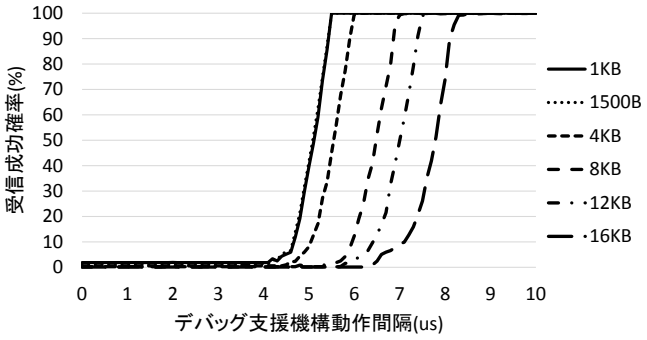

図 12 UDP 受信プログラムにおける受信成功確率

間隔とパケットサイズならば，パケットを受信できるかを測定する．また，デバッグ支援 OS からデバッグ対象 OS 上のプログラムまでで実現できている通信量を測定する．具体的な方法は，6.5.1 節とほぼ同様である．カウンタをインクリメントする箇所が 6.5.1 節では NIC ドライバがパケットを受信した時であったのに対し，本測定では UDP の受信プログラムがパケットを受信した時であることが差分である．

6.6.2 測定結果

測定結果を図 12 に示し，読み取れることを以下に示す．

(1) どのパケットサイズでもある時点まではほとんど受信に失敗している．
(2) ある時点から一次関数的に受信成功確率が増加している．
(3) 受信成功確率が増加し始めるある時点は，パケットサイズが増加するに連れて増大している．
(4) 受信成功確率が一次関数的に増加する際，パケットのサイズに関わらず，傾きは一定である．
(5) 受信成功確率が 0 回になるデバッグ支援機構の動作間隔以上に間隔を増加させても受信成功確率は 100%となる．

6.7 考察

6.7.1 測定 1

測定結果である 6.4.2 項から，現在の測定環境では IPI の送信間隔が約 1us 以上であれば連続で IPI を送信した際に，全ての IPI に割り込みハンドラが動作できることが分かる．したがって，本実験環境下では，連続で割り込みを発生させる際，最低でも 1us 以上の間隔は必要であると言える．

6.7.2 測定 2

測定結果である 6.5.2 項から，連続で割り込みを発生させた際，パケットのサイズを増加させるほど全てのパケットを処理することにかかる時間が増加することが分かる．また，NIC ドライバで測定した受信成功回数，パケットのサイズ，および 1 サイクルにかかった時間から各パケットにおける通信量を算出した．各パケットサイズにおける初めて受信成功確率が 100%になった際のデバッグ支援機構の動作間隔，1 サイクルにかかった時間，総データ量，および通信量を表 2 に示す．

表 2 NIC ドライバにおけるパケットのサイズと割り込み間隔の関係

size	interval	time	amount of data	throughput
1KB	4.3us	21689us	5000KB	1.7Gbps
1500B	4.4us	22203us	7324KB	2.5Gbps
4KB	4.6us	23195us	20000KB	6.5Gbps
8KB	5.5us	27788us	40000KB	10.9Gbps
12KB	5.9us	29770us	60000KB	15.3Gbps
16KB	6.4us	32491us	80000KB	18.7Gbps

6.7.3 測定 3

測定結果である 6.6.2 項から，連続で割り込みを発生させた際，NIC ドライバでの結果と比べて，より多くの時間がかかっていることが分かる．このことから，NIC ドライバですべてのパケットを処理できていても，プログラムまでパケットが届くにはより多くの時間がかかることが分かる．また，UDP の受信プログラムで測定した受信成功回数，パケットのサイズ，および 1 サイクルにかかった時間から各パケットサイズにおける通信量を算出した．各パケットサイズにおける初めて受信成功確率が 100%になった際のデバッグ支援機構の動作間隔，1 サイクルにかかった時間，総データ量，および通信量を表 3 に示す．

表 3 UDP 受信プログラムにおけるパケットのサイズと割り込み間隔の関係

size	interval	time	amount of data	throughput
1KB	5.5us	27677us	5000KB	1.3Gbps
1500B	5.5us	27700us	7324KB	2.0Gbps
4KB	6.1us	30705us	20000KB	4.9Gbps
8KB	7.7us	38649us	40000KB	7.8Gbps
12KB	9.4us	47241us	60000KB	9.6Gbps
16KB	11.1us	55789us	80000KB	10.9Gbps

6.8 実際の NIC との比較

上記の結果と考察から, NIC ドライバで最大約 18.7Gbps, デバッグ対象 OS 上で動作する UDP 受信プログラムで最大約 10.9Gbps のスループットを実現できることが分かった．PCI Express 1.1 の 1 レーンの転送量が 2.5Gbps であることから，実際の NIC が実現できる最大のスループットは 2.5Gbps であると言える．本デバッグ支援環境で実現できるスループットと実際の NIC が実現できるスループットを比較すると，本デバッグ支援環境は実際の NIC のスループットを大きく超えたスループットを実現できていることがわかる．このことから，本デバッグ支援環境を用いることで現在は実現できていない高スループットの NIC

のエミュレートができると考えられる．高スループットのNIC やバスが開発されたと想定し，これらに対応するドライバを開発できる．

7. おわりに

本稿では Mint を用いた NIC ドライバの割り込みデバッグ手法について述べた．まず既存研究である VM を用いた OS の割り込みデバッグ手法と問題点について述べた．次に Mint と Mint を用いた割り込みデバッグ手法について述べた．次に，Mint を用いた NIC ドライバの割り込みデバッグ環境の方針，課題，対処，および必要な機能について述べた．そして，Mint を用いた割り込みデバッグ環境の実装について述べた．最後に Mint を用いた割り込みデバッグ環境の評価について述べた．

Mint を用いた NIC ドライバの割り込みデバッグ環境では，実割り込みを発生させられる環境と，NIC を用いずパケットを授受する環境を提供する．設計の課題として，割り込み間隔と回数の調整，パケットの作成，パケットの格納，受信バッファ状態の更新，割り込み契機の変更，割り込みハンドラの作成，および受信バッファの作成を示した．これらの対処として，割り込みジェネレータの作成，デバッグ支援機構の作成，IPI の送信，および NIC ドライバの改変を示した．

本研究では NIC のパケット受信処理に対する NIC ドライバの割り込み処理をデバッグ対象とした．これを実現する機能として，割り込みジェネレータ，パケットの作成，受信バッファへのパケットの格納，受信バッファ状態の更新，IPI の送信，割り込みハンドラ，および受信バッファの作成について示した．これらの機能の内，割り込みジェネレータはデバッグ支援 OS 上で動作する AP として実装し，割り込み情報を指定してデバッグ支援機構に通知することを示した．パケットの作成，受信バッファへのパケットの格納，受信バッファ状態の更新，および IPI の送信についてはデバッグ支援機構の機能として動作するもので，システムコールとして実装することを示した．割り込みハンドラ，および受信バッファの作成については NIC ドライバを改変し，実現することを示した．実装した環境を動作させることにより，ネットワーク層で破棄されたパケットを確認したため，本環境において割り込み処理を再現できたことを確認した．また，実装した本環境において，2400ns 以上の間隔ならば，連続して割り込み処理を再現できることを示した．

参考文献

[1] 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書，Vol. 110, No. 278, pp. 29–34 (2010).

[2] 宮原俊介，吉村　剛，山田浩史，河野健二：仮想マシンモニタを用いた割込み処理のデバッグ手法，情報処理学会研究報告，Vol. 2013-OS-124, No. 6, pp. 1–8 (2013).

[3] Samuel, T.K., George, W.D. and M.C., P.: Debugging operating systems with time-travelling virtual machines, *Proceedings of The USENIX Annual Technical Conference*, pp. 1–15 (2005).

[4] Jim, C., Tal, G., Peter and M.C.: Decoupling dynamic program analysis from executionin virtual environments, *USENIX 2008 Annual Technical Conference*, pp. 1–14 (2008).

[5] 川崎　仁，追川修一：SMP を利用した Primary/Backup モデルによるリプレイ環境の構築，情報処理学会研究報告，Vol. 2010-OS-113, No. 12, pp. 1–8 (2010).

[6] 北川初音，乃村能成，谷口秀夫：Mint: Linux をベースとした複数 OS 混載方式の提案，情報処理学会研究報告，Vol. 2013-OS-126, No. 17, pp. 1–8 (2013).