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) の多機能化により OS のテストが重要になっている．しかし，OS はハードウェアの動作に左右される機能が多く，OS 単体でのテストが困難である．困難なテストの 1 つに，割り込み処理に関するテストがある．割り込み処理は再現が困難であり，このテストもまた困難である．そこで，Mint を用いた OS の開発支援手法が提案されている．Mint は 1 台の計算機上で複数の OS が独立して走行する方式である．本研究では，NIC ドライバを対象とした割り込み処理の開発支援環境を実現した．Mint を用いて，NIC ドライバのパケット受信割り込み処理を再現する．この手法により，NIC のパケット受信割り込み処理を再現し，テストを支援する．このテスト支援環境を用いて，NIC ドライバのテストを行ったところ，最大で約 30Gbps を実現できることを示した．

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割り込みの再現が可能になる．

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

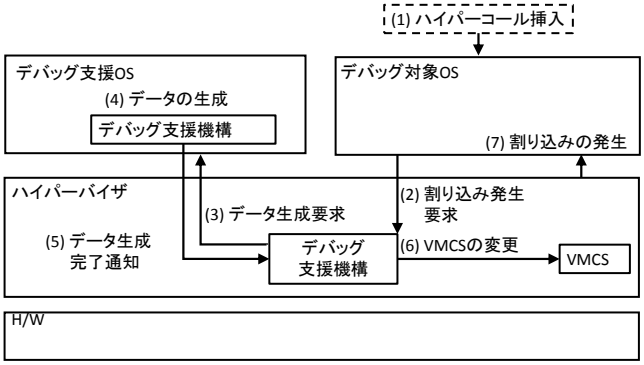

図 1 割り込み挿入法の処理流れ

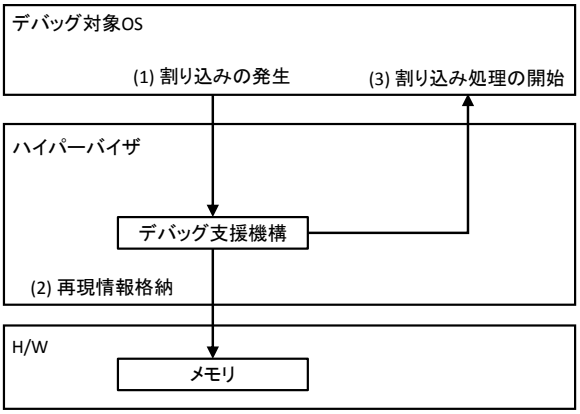

図 2 ロギングの処理流れ

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．

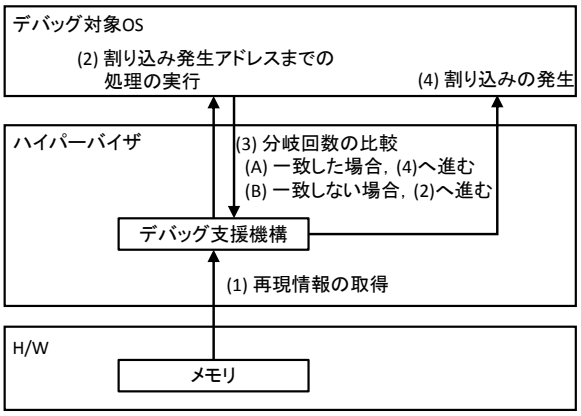

図 3 リプレイの処理流れ

(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

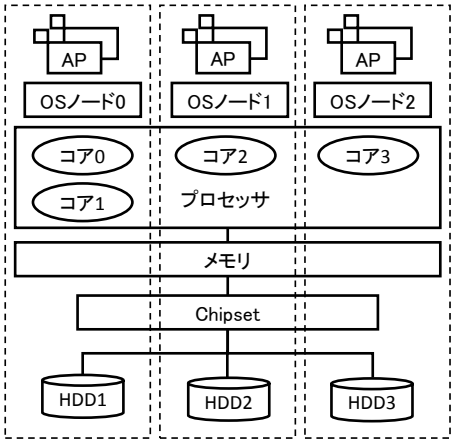

図 4 Mint の構成

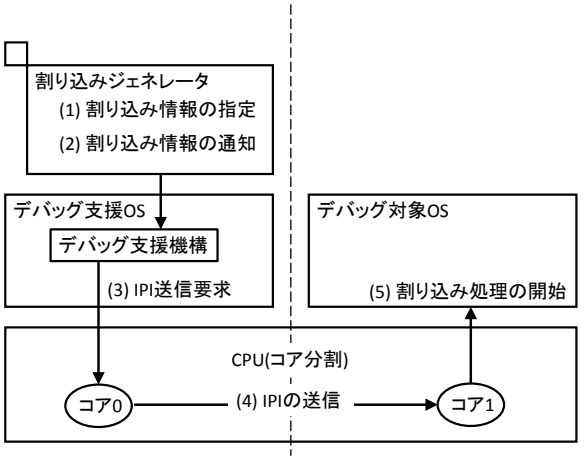

図 5 Mint を用いたデバッグ支援環境の処理流れ

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

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．本稿では主に (方針 1) である実割り込みの再現について設計する．

4.2 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．

(設計方針 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．

(設計方針 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．このため，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．

(設計方針 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．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．

4.3 NIC ドライバのパケット受信の流れ

本研究では 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．

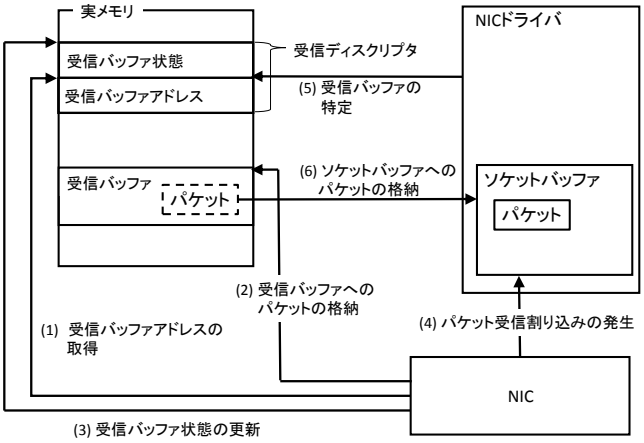

図 6 NIC ドライバのパケット受信処理流れ

4.4 課題

設計にあたっての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 (設計方針 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 (設計方針 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 (設計方針 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 (設計方針 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 (設計方針 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改変 (設計方針 2)
NIC(ハードウェア) を用いないため，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ように割り込みハンドラを改変する必要がある．

(課題 7) 受信バッファの作成 (設計方針 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割り込み情報の指定 (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) 割り込みジェネレータ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，デバッグ支援機構を動作させ，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 により動作するように 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，受信バッファからのパケットの取得とソケットバッファへのパケット格納機能を持つ．

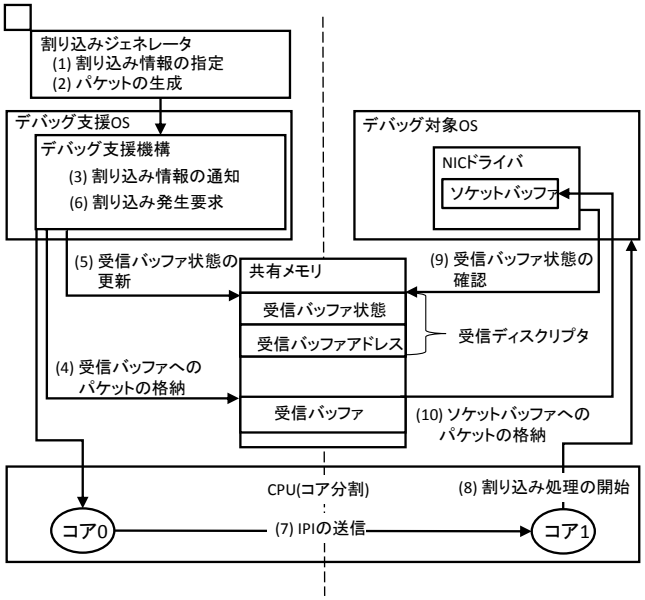

図 7 NIC ドライバのデバッグ支援環境の処理流れ

(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．

5. 実装

4 章の対処を元に，デバッグ支援環境を実装した．実装したデバッグ支援環境の処理流れを図 7 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パケットを作成する．本実装では，作成するパケットは UDP パケットを含んだ EthernetFrame である．
(3) 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と (2) で作成したパケットを引数とする．
(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．

(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．

6. 評価

6.1 評価項目

実装したデバッグ支援環境を以下の項目で評価する．

(評価 1) 実現可能な割り込み間隔の評価
(評価 2) 割り込み間隔の精度の評価
(評価 3) 本環境の有用性の評価

(評価 1) では，本環境を用いることで，どの程度の短い間隔で割り込みを発生できるかを測定し，実割り込みの再現について評価する．送信処理にはパケットをメモリに複写する処理が含まれる．このため，パケットサイズ毎に送信処理時間が長大し，実現できる送信間隔も長大すると考えられる．

(評価 2) では，デバッグ支援 OS で割り込み間隔を指定して連続で割り込みを発生させた際，デバッグ対象 OS において，どの程度の精度で指定した割り込み間隔を実現できているかを評価する．

(評価 3) では，本環境を用いた際の有用性を示すため，NIC ドライバの性能測定を行う．本環境を用いて，送信間隔を指定して連続でパケットを送信した際，その間隔ならば，どの程度の確率でパケットを受信できるかを測定する．また，どの程度の通信量を実現できているかを測定する．

6.2 実現可能な送信間隔の評価

本デバッグ支援環境がどの程度の短い間隔で割り込みを発生可能かを評価する．割り込みは IPI の送信によって発生するため，IPI の送信間隔は割り込みの発生間隔といえる．IPI の送信はパケットの送信処理の最後に行われるため，1 回の送信処理時間が割り込み発生間隔の最小値である．ここで，パケットの送信処理とは図 7 の (4)〜(7) である．このため，送信処理にかかる時間を測定し，実現可能な割り込み間隔の最小値を評価する．

送信処理にはメモリ複写処理が含まれており，パケットのサイズによって送信処理が変わると考えられる．したがって，3 つのパケットサイズにおける送信処理時間を評価する．具体的には，各パケットサイズで 3000 回送信処理を測定し，1000 回目〜2000 回目の処理時間の平均を取ることで，処理時間とした．これは，処理の開始と終了には不確定な要素により，処理時間が大きく変動する可能性があるためである．測定に用いるパケットサイズは，以下の 3 つである．

(1) 1.5KB(MTU のサイズ)

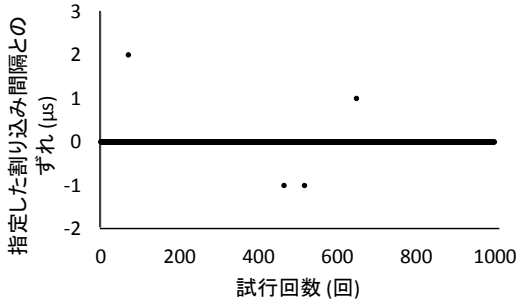

図 8 38 μ s の間隔を指定した際の割り込み発生間隔のずれ

(2) 8KB(受信バッファの半分のサイズ)
(3) 16KB(受信バッファのサイズ)

結果を表 1 に示し，以下で説明する．パケットのサイズが増加するに連れ，送信処理時間が増加していることが分かる．これは，送信処理中にメモリ複写処理が含まれるためである．また，本デバッグ支援環境を用いて，連続でパケットを送信しようとした際に実現できる最短の時間が表 1 である．したがって，表 1 に示した時間以上の割り込み間隔を指定できる．

表 1 各パケットサイズにおける送信処理時間

パケットサイズ (KB)	処理時間 (μ s)
1.5	0.205
8	1.462
16	3.664

6.3 割り込み間隔の精度の評価

本環境を用いて連続で割り込みを発生させた際，デバッグ対象 OS でどの程度の精度で指定した割り込み間隔を実現できているかを評価する．

実際に一方の計算機から他方の計算機へ NIC を用いて連続でパケットを送信すると，約 38 μ s の間隔で割り込みが発生していることを確認した．この際，常に割り込み間隔は安定せず，試行毎に 38 ± 2 μ s ほどぶれていた．また，稀に大きな外れ値があり，一定の間隔では割り込みが発生していないことを確認した．

そこで，割り込み間隔として 38 μ s を指定し，1000 回連続でデバッグ対象 OS に割り込みを発生させた際，割り込みハンドラが動作する間隔を測定し，それぞれの間隔が 38 μ s とどれだけ差があるかを測定し，評価する．

結果を図 8 に示し，以下で説明する．ほとんどの場合，指定した間隔である 38 μ s で割り込みが発生していることが分かる．1000 回の試行で 38 μ s から外れたものは 4 回であり，一番大きな外れ値が+2 μ s である．38 μ s というスケールから見ると，この程度のズレは許容範囲内であると言える．したがって，本環境を用いることで，安定して指定した間隔で割り込みを発生させられることがわかる．

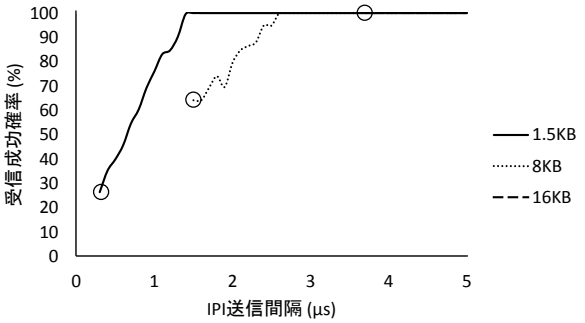

図 9 送信処理動作間隔と NIC ドライバにおけるパケット受信成功率の関係

6.4 本環境の有用性の評価

6.4.1 NIC ドライバの性能測定

本デバッグ支援環境の有用性を示すため，本環境を用いて NIC ドライバの性能を測定する．以下の 2 つの項目で性能測定を行う．

(1) 各パケットサイズにおける NIC ドライバで処理可能な割り込み間隔の測定
(2) 各パケットサイズにおける NIC ドライバで実現可能な通信量の測定

これらの項目を測定することにより，NIC ドライバの処理できる限界を調査できる．これにより，NIC ドライバがどの程度高速な NIC に対応できるかがわかる．

6.4.2 NIC ドライバで処理可能な割り込み間隔の測定

指定した間隔で送信処理を 5000 回動作させ，NIC ドライバでのパケット受信成功率を求めることを 1 サイクルとし，割り込み間隔を増加させながら複数サイクル行う．これにより，割り込み間隔がどの程度ならば受信成功率が 100%になるかを測定する．受信処理にはメモリ複写処理が含まれるため，パケットサイズによって 100%になる間隔が増加すると考えられる．このため，この試行を複数のパケットサイズで行う．測定に使用するパケットサイズは 6.4.3 節と同じ 3 つを用いる．

結果を図 9 に示し，以下で説明する．なお，図中の丸印は各パケットサイズで実現可能な送信間隔の最小値 (表 1) を示しており，これ以下の間隔は実現できないためこれ以上の間隔の結果を示している．結果から，送信間隔を増加させるに連れ，受信成功率が 1 次関数的に増加していることが分かる．また，パケットのサイズが増加すると，受信成功率が 100%になるまでの間隔が長大している．これは，パケット受信割り込み処理中にメモリの複写が行われるためだと考えられる．

すべてのパケットを受信可能な間隔が送信処理の最小値に比べて大きいのは，受信処理にはソケットバッファをドライバよりも上位のレイヤに送信する処理が含まれており，これに長い時間を費やすためである．

6.4.3 NIC ドライバで実現可能な通信量

連続でパケットを送信した際，各パケットサイズ毎に NIC ドライバで実現できる最大の通信量を測定した．具体的には，6.4.2 項と同様の方法で測定を行った際の初めてパケット受信成功率が 100%となった際の送信処理 5000 回にかかった時間とパケットのサイズから各パケットの通信量を算出した．また，同様に NIC ハードウェアを用いて NIC ドライバで処理できる最大の通信量を測定した．この際，パケットサイズは MTU である 1.5KB とした．これらを比較し，評価する．

結果を表 2 に示し，以下で説明する．表 2 から実 NIC を大きく超える通信量を実現できていることがわかる．また，最高で 30Gbps を実現できており，現在は開発されていない高速な NIC をシミュレートできると考えられる．メモリのみを用いて通信を行なっているため，このような膨大な通信量を実現できる．測定を行った計算機のメモリ帯域幅は約 130Gbps であり，受信処理にはメモリ複写以外に長い時間を費やす処理が含まれているため，これは妥当な結果であると言える．

表 2 各パケットサイズにおける実現可能な通信量

パケットサイズ (KB)	通信量 (Gbps)
1.5(実 NIC)	0.92
1.5	6.3
8	22.7
16	30.6

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．

実装したデバッグ支援環境を用いて，連続で割り込みを発生させた際，指定した間隔を守って，割り込みを発生させられることを示した．また，本デバッグ支援環境が NIC ドライバの性能評価に有用であることを示した．

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).