Mint オペレーティングシステムを用いた NIC ドライバの開発支援手法の実現

藤田 将輝[1]　乃村 能成[1]　谷口 秀夫[1]

概要：OS の機能の中でも，割込処理は非同期的に発生するため，処理の再現性がなく，開発が困難である．仮想計算機を用いて割込をエミュレートする開発手法がこれまで提案されているが，ハイパーバイザのオーバヘッドにより，短い間隔や一定間隔で発生する割込を再現できない．一方，我々が研究開発している Mint オペレーティングシステムでは，仮想化を用いずに複数の OS を動作させることで，上記の問題を解決する．具体的には，Mint 上で動作する 2 つの OS の一方を擬似的なデバイスとし，その擬似デバイスから高速で一定周期の割込を発生させる．本論文では，提案手法を割込処理が多発する NIC ドライバに適用し，NIC ドライバの開発支援に有用であることを示す．

キーワード：仮想化，割込，開発

1. はじめに

Operating System (以下，OS) の多機能化にともなって，OS のテストやデバッグに多くの工数を要するようになっている [1]．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 つに割込処理がある．割込処理の再現をする方法として，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 台の計算機上で開発を支援する機構 (以下，開発支援機構) と開発対象の OS(以下，開発対象 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，開発支援機構を開発対象 OS の外部に実装できる点である．これにより，開発支援機構が開発対象 OS の影響を受けない．開発支援機構が開発対象 OS へ割込を挿入させたり，開発支援機構が開発対象 OS の動作を再現したりすることで，開発を支援する．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 を用いた開発支援環境では一定間隔で発生する割込や，短い間隔で発生する割込のように，処理負荷が影響する割込処理の再現が困難である．

そこで，マルチコア CPU の 1 コアを開発対象 OS の外部に設置し，このコアから開発対象 OS へ割込を発生させることにより問題を解決する．これを可能にする手段として，Multiple Independent operating systems with New Technology (以下，Mint)[2] が研究開発されている．Mint はマルチコア CPU を搭載した計算機上で仮想化用いずに複数の OS を動作できる．

本研究では，提案手法を用いて非同期的な割込が頻繁に発生する NIC ドライバを対象とした開発支援環境の実装について述べる．これにより，Mint における割込処理の開発支援環境で，NIC ドライバのテストが行えることを示す．

2. 関連研究

2.1 VM を用いた開発支援機構

OS の割込に関するテスト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．既存研究は主に，バグを再現することでデバッグを支援するデバッグ支援環境として研究されている．VM を用いた割込開発支援環境は大きく分けて 2 つある．割込挿入法 [3] とロギング/リプレイ法 [4][5][6][7] である．これらの概要について以下で説明する．

割込挿入法

割込挿入法は開発対象 OS の他に，開発支援機構として，ハイパーバイザと開発を支援する OS(以下，開発支援 OS) が走行する．ユーザが開発対象 OS の割込を挿入したいコード位置にハイパーコールを挿入することで，割込を発生させる．この際，割込は 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

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

© 1960 Information Processing Society of Japan

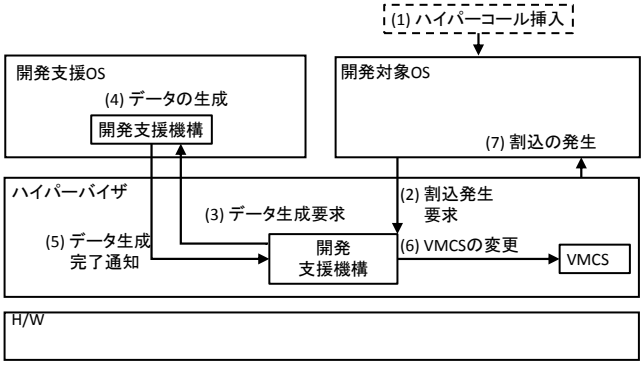

図 1 割込挿入法の処理流れ

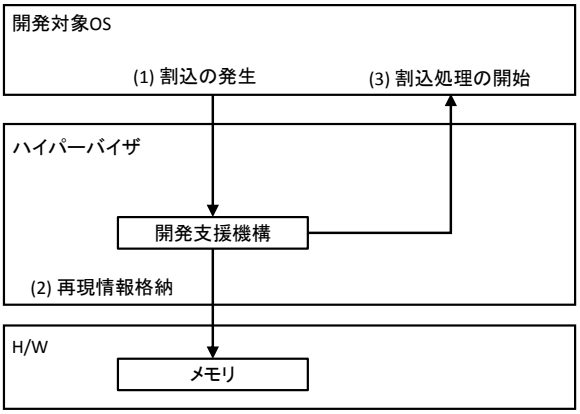

図 2 ロギングの処理流れ

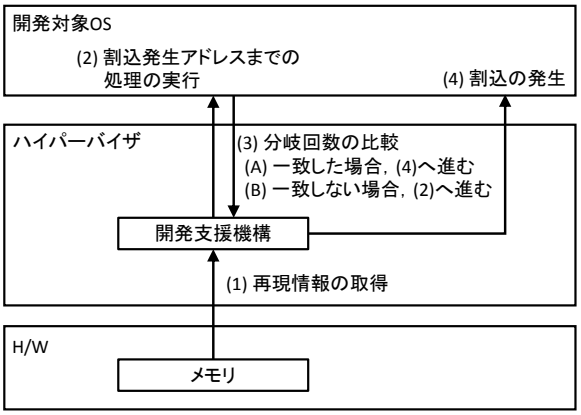

図 3 リプレイの処理流れ

る．割込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) ユーザが割込を挿入したい位置にハイパーコールを挿入する．この際，割込の種類とデータを指定する．

(2) 開発対象 OS に挿入したハイパーコールにより，開発対象 OS がハイパーバイザの開発支援機構へ割込発生要求を行う．その後，開発対象 OS の処理を中断し，ハイパーバイザへ処理が遷移する．

(3) ハイパーバイザの開発支援機構が開発支援 OS の開発支援機構へ割込に必要なデータの生成要求を行う．

(4) 開発支援 OS の開発支援機構が割込に必要なデータを生成する．

(5) 開発支援 OS の開発支援機構がハイパーバイザの開発支援機構へデータの生成完了を通知する．

(6) ハイパーバイザの開発支援機構が 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開発対象 OS へ処理が遷移するとき割込が発生する．

(7) 開発対象 OS へ処理が遷移し，割込が発生する．

割込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込処理の開発手法 [3] がある．これは仮想マシンモニタが開発対象 OS に仮想的な割込を発生させるものである．

ロギング/リプレイ法

ロギング/リプレイ法は開発対象 OS の他に，開発支援機構として，ハイパーバイザが走行する．この手法は開発対象 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，バグを再現し，開発を支援する．ロギングの処理流れを**図 2** に示し，以下で説明する．

(1) 開発対象 OS に割込が発生すると，処理を中断し，ハイパーバイザに処理が遷移する．

(2) ハイパーバイザの開発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開発対象 OS へ処理が遷移し，開発対象 OS が中断していた割込処理を再開する．

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) ハイパーバイザの開発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開発対象 OS が割込が発生するアドレスまで命令を実行する．

(3) ハイパーバイザの開発支援機構が再現情報の分岐回数と，現在の開発対象 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) 開発対象 OS へ割込が発生する．

ロギング/リプレイ法を用いた既存研究として TTVM[4]，Aftersight[5]，Sesta[6]，および LoRe[7] がある．TTVM は再現情報に加え，開発対象 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．LoRe はログのサイズと再現時間の削減を行っている．

© 1960 Information Processing Society of Japan

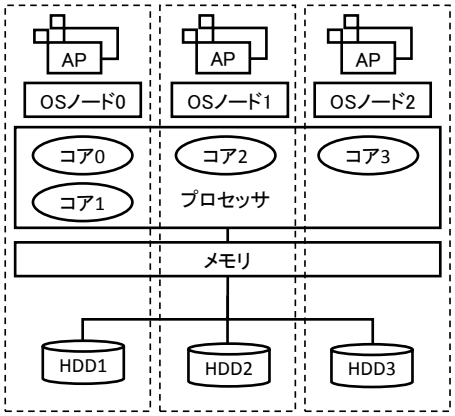

図 4 Mint の構成

2.2 問題点

既存手法の割込挿入法とロギング/リプレイ法のそれぞれの問題点について以下で説明する．

(割込挿入法) 任意の間隔で発生する複数割込の挿入が困難

割込挿入法では割込を発生させる際，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込を発生させる．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，複数の割込を 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．この間隔をユーザが調整するのは困難である．したがって，から割込挿入法では任意の間隔での割込挿入が困難である．

(ロギング/リプレイ法) 実時間を反映したロギングが困難

ロギング/リプレイ手法は，ロギングにおける開発対象 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，実際に発生した割込間隔と再現情報の割込間隔には処理負荷分のずれができる．したがって，実時間を反映したロギングが困難であり，再現する割込は実際に発生する割込間隔よりも長い間隔となってしまう．

これらの問題点から，既存手法では短い間隔で複数割込を発生させるようなストレステストやパフォーマンスチューニングのようなテストには使用できないと考えられる．短い割込や任意の間隔の割込を発生させるためには，開発対象 OS が開発支援機構の処理負荷の影響を受けない環境が必要である．また，実際の割込処理の再現をするため，これらの環境は実際の割込処理と同様の挙動をする必要がある．

3. Mint を用いた開発支援環境

3.1 Mint オペレーティングシステム

Mint オペレーティングシステムは 1 台の計算機上で仮想化を用いずに複数の OS を動作できる方式である．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．

このようにして Mint ではマルチコアプロセッサ CPU のコアを分割し，複数の Linux を同時に走行できる．Mint を用いることで，開発対象の OS の外に開発支援 OS を動作させ，開発支援 OS から開発対象 OS へ割込を発生させることで開発を支援する環境を提案する．

3.2 方針

2.2 節に示した VM を用いた割込開発支援手法の問題点を解決する手段としての Mint を用いた開発支援環境の方針について以下に示し，説明する．

(方針 1) 実際に発生する割込間隔を実現する環境の提供
ストレステスト等の短い割込を連続で発生させるようなテストを実施するには，実際に発生する割込間隔程度の間隔で割込を発生できる環境が必要である．しかし，VM を用いた既存研究では 2.2 節で述べた問題により，短い間隔の割込の発生が困難である．そこで Mint を用いた割込処理の開発支援環境では，開発対象 OS へ実際に発生する割込間隔程度の短い間隔で割込を発生できる環境を提供する．

(方針 2) 任意のパタンで割込を発生させる環境の提供
テストによっては，短い周期や長い周期で複数の割込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 2.2 節で述べた問題により，任意の間隔での割込発生が困難である．そこで Mint を用いた割込処理の開発支援環境では，開発対象 OS へ任意の間隔で割込を発生できる環境を提供する．

3.3 構成と処理流れ

Mint を用いた開発支援環境について**図 5** に示し，説明する．Mint 上で開発支援 OS を OS ノード 0，開発対象 OS を OS ノード 1 として動作させる．開発支援 OS はコア 0 を占有し，割込管理アプリケーション (以下，割込管理 AP) と開発支援機構を持つ．開発対象 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，割込を発生させる．

© 1960 Information Processing Society of Japan

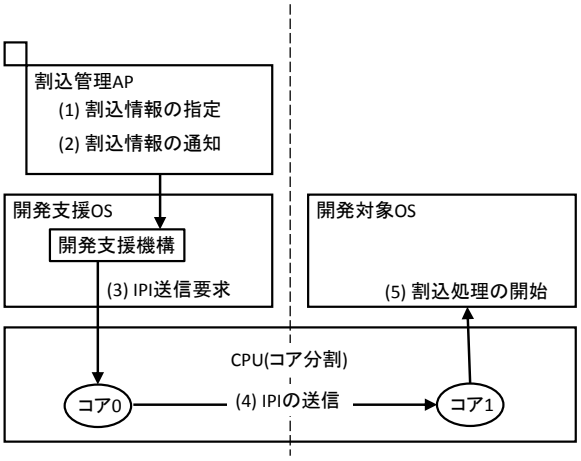

図 5 Mint を用いた開発支援環境の処理流れ

(1) 開発支援 OS 上で動作するアプリケーションである割込管理 AP を用いてユーザが割込情報を指定する．
(2) 割込管理 AP がシステムコールを用いて開発支援機構を呼び出す際，指定した割込情報を開発支援 OS の開発支援機構に通知する．
(3) 開発支援機構がコア 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 を受信すると割込処理が開始する．

3.4 Linux 改変による割込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 [8]．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 における割込処理は，改変前の Linux の割込処理と同等であるといえる．このため，Linux の開発に有用であると考えられる．

4. NIC ドライバの開発支援環境の設計

4.1 目的

割込処理の 1 つに，デバイスドライバの割込処理がある．NIC では頻繁に通信を行なっているため，割込処理も頻繁に行われている．また，複雑化するデバイスドライバの開発を支援する手法が重要になっている．デバイスドライバを開発対象とした研究として SymDrive[9] がある．これはデバイスを用いずにドライバのコードを網羅的に実行することでバグを発見し，開発を支援する．しかし，SymDrive では任意に割込を発生できないため，ストレステストやパフォーマンスチューニングのようなテストには使用できない．また，NIC の高速化により NIC ドライバやプロトコル処理の高速化させる必要があり [10]，これらのテストが重要になっている．そこで，NIC ドライバを対象とした Mint を用いた割込開発支援環境を構築し，割込処理を再現できる環境を実現する．これにより，Mint における割込処理の開発支援環境で，割込処理を再現する．

4.2 設計方針

Mint を用いた NIC ドライバの割込開発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) 任意の間隔と回数の割込発生
テストによっては任意の間隔の割込を任意の回数分発生させる必要がある．このため，本開発支援環境では発生させる割込の間隔と回数をユーザが指定可能な環境を提供する．

(設計方針 2) NIC の動作を開発支援 OS が再現
NIC ドライバの割込処理の開発を対象とするため，ハードウェア (NIC) の動作は考慮しない．したがって，NIC を用いずに割込開発支援環境を構築する．このため，開発支援 OS が NIC を再現することとする．これにより，NIC を用いずに NIC ドライバの割込開発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 のパケット受信割込処理を再現するため，開発対象 OS でパケットを受け取る必要がある．開発対象 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．本研究ではテスト対象のデバイスの動作を考慮しないため，入出力デバイスは用いない．そこで，開発支援 OS と開発対象 OS 間の共有メモリを用いる．共有メモリ上に NIC の受信バッファを配置することにより，開発支援 OS から開発対象 OS の NIC ドライバへパケットを受け渡す．

4.3 課題

設計にあたって，4.2 節に挙げた各設計についての課題を以下に示し，説明する．

(設計方針 1)

(課題 1) 割込間隔，回数の調整
任意の間隔と回数で割込を発生させるため，割込の間隔を調整し連続で割込を発生させる必要がある．これらの情報をユーザが指定可能にする必要がある．

(設計方針 2)

(課題 2) パケットの作成，格納
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し，受信バッファに格納する必要がある．

(課題 3) NIC の状態の更新
NIC がパケットを受信バッファに格納する際，NIC は受信バッファ状態を書き換え，受信済み状態にする．受信バッファ状態とは受信バッファがパケット

© 1960 Information Processing Society of Japan

を受信しているか否かの状態である．この状態は受信ディスクリプタという受信バッファを管理する構造に含まれている．本環境では NIC ハードウェアを用いないため NIC を用いずに受信バッファ状態を書き換える必要がある．

(課題 4) 割込処理の発生

本環境では NIC ハードウェアを用いないため，NIC の機能である OS へ割込を発生させる機能を再現する必要がある．また，発生させた割込に NIC ドライバの割込ハンドラが反応可能にする必要がある．

(設計方針 3)

(課題 5) 受信バッファの作成

開発支援 OS が共有メモリにパケットを配置し，開発対象 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．

4.4 対処

課題への対処を以下に示し，説明する．また，各対処の通番は課題と対応している．

(対処 1) 割込管理 AP による割込情報の指定

割込間隔と回数をユーザが指定できるようにするため，開発支援 OS 上にこれらの情報が指定できる割込管理 AP を実装する．指定した間隔と回数で割込を発生させるシステムコールを発行する．

(対処 2) パケットの作成，格納

割込管理 AP でパケットを作成し，作成したパケットをシステムコールにより開発支援機構に渡す．その後，開発支援機構が受信バッファにパケットを格納する．

(対処 3) NIC の状態の更新

受信バッファ状態を開発支援 OS が書き換え可能にするため，受信ディスクリプタを共有メモリに配置し，開発支援 OS と開発対象 OS の両 OS で参照可能にする．

(対処 4) 割込契機として IPI を使用

割込の契機としてコア間割込である IPI を用いる．開発支援 OS が占有するコアから開発対象 OS が占有するコアへ IPI を送信することで割込処理を発生させる．また，IPI により NIC ドライバの割込ハンドラが反応するように改変を加える．この際の改変は割込ハンドラが動作するまでの流れを変更するものであり，ハンドラの処理自体に影響はない．

(対処 5) 共有メモリへ受信バッファの作成

共有メモリを用いて開発支援 OS と開発対象 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．

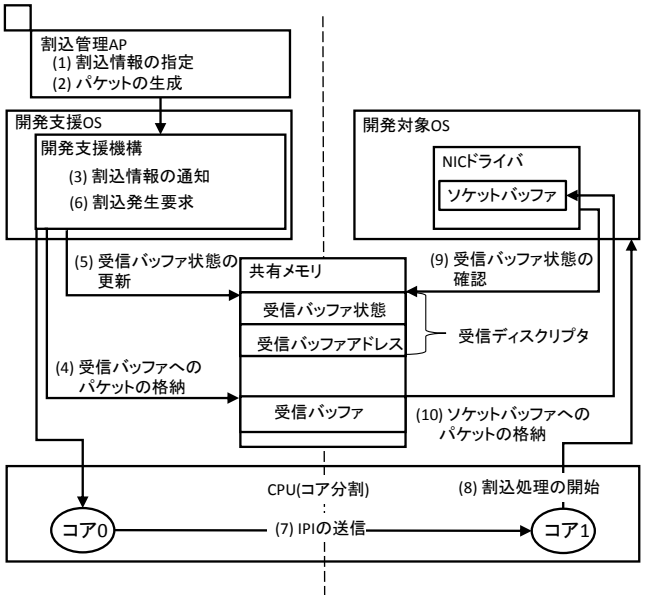

図 6 NIC ドライバの開発支援環境の処理流れ

5. 実装

4 章の対処を元に，開発支援環境を実装した．実装した開発支援環境の処理流れを**図 6** に示し，以下で説明する．

(1) ユーザが割込管理 AP を用いて割込情報を指定する．
(2) 割込管理 AP 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 Ethernet-Frame である．
(3) 割込管理 AP が開発支援 OS の開発支援機構をシステムコールを用いて呼び出す．この際 (1) で指定した割込情報と (2) で作成したパケットを通知する．
(4) 開発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) 開発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) 開発支援機構がコア 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，開発対象 OS の割込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

実装した開発支援環境を以下の項目で評価する．

(評価 1) 実現可能な割込間隔の評価

(評価 2) 割込間隔の精度の評価

(評価 3) 実テストへの応用評価

© 1960 Information Processing Society of Japan

(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込を発生できるかを測定し，実現可能な割込間隔について評価する．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，開発支援 OS で割込間隔を指定して連続で割込を発生させた際，開発対象 OS において，どの程度の精度で指定した割込間隔を実現できているかを評価する．

(評価 3) では本環境を用いて 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，この測定結果からどの程度の通信量を実現できているかを算出する．

6.2 実現可能な送信間隔の評価

本開発支援環境がどの程度の短い間隔で割込を発生可能かを評価する．割込は IPI の送信によって発生するため，IPI の送信間隔は割込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込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 6 の (4)〜(7) である．このため，送信処理にかかる時間を測定し，実現可能な割込間隔の最小値を求める．

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 1000 回送信処理を測定し，平均時間を取ることで，処理時間とした．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)
(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．パケットのサイズの増加にともなって，送信処理時間が長大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．表 1 の値は本開発支援環境を用いて，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であるといえる．したがって，表 1 に示した時間以上の送信間隔を実現できる．

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

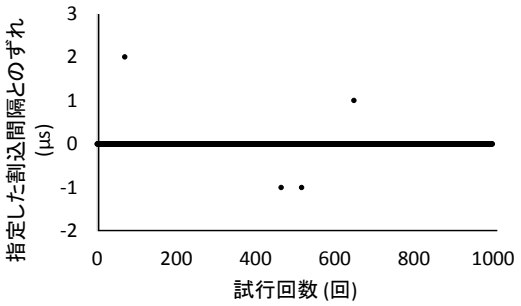

図 7 38 μ s の間隔を指定した際の割込発生間隔のずれ

6.3 割込間隔の精度の評価

本環境を用いて連続で割込を発生させた際，開発対象 OS でどの程度の精度で指定した割込間隔を実現できているかを評価する．

評価にあたり実計算機 2 台を LAN ケーブルで接続し，NIC を用いて連続でパケットを送信した際，約 38 μ s の間隔で割込が発生していることを確認した．この際，常に割込間隔は安定せず，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，一定の間隔では割込が発生していないことを確認した．

そこで，本環境を用いて割込間隔を 38 μ s に指定し，1000 回連続で開発対象 OS に割込を発生させた際，割込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，精度について評価する．

結果を**図 7** 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 と差がなく，指定した間隔で割込を発生でき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 から見ると，全てのずれは± 6.25%以内に収まっている．

6.4 実テストへの応用評価

6.4.1 NIC ドライバの性能測定

本開発環境を用いて実際に NIC ドライバのテストを行う．以下の 2 つの項目で性能評価テストを行う．

(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込間隔の測定
(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速度の測定

これらの項目を測定することにより，NIC ドライバの処理性能を調査する．これにより，本環境が高速な NIC を擬似することにより NIC ハードウェアを用いずに開発対象ドライバの処理性能をテストできることを示す．本環境を用いてテストを行うことにより，開発対象ドライバがどの程度高速な NIC に対応できるかが分かる．

6.4.2 NIC ドライバで処理可能な割込間隔の測定

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

© 1960 Information Processing Society of Japan

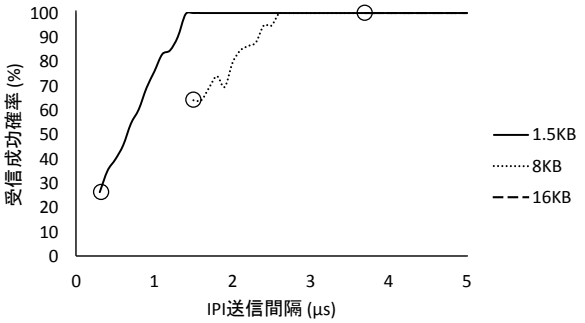

図 8 送信処理動作間隔と NIC ドライバにおけるパケット受信成功率の関係

し，割込間隔を増加させながら複数サイクル行う．これにより，割込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，パケットサイズの増加にともなって 100%になる間隔が長大化すると考えられる．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 節と同じ以下の 3 つのものを用いる．

(1) 1.5KB(MTU のサイズ)
(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**図 8** 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，送信間隔の増加にともなって，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，パケットのサイズの増加にともなって，受信成功率が 100%になるまでの間隔が長大化している．これは，パケット受信割込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いるためである．

6.4.3 NIC ドライバで実現可能な通信速度

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速度を算出した．具体的には，6.4.2 項と同じ方法の測定において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．具体的には，2 台の計算機を LAN ケーブルで接続し，一方の計算機から他方の計算機へパケットを連続で送信した際の通信速度を測定した．測定において，どちらの場合もパケットサイズは 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速度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，実 NIC より高速な NIC をエミュレートできていることが分かる．

このことから，新規 NIC ドライバ開発において開発対象ドライバを高速な NIC に対応させる際に本環境を用いることで，NIC を用いずに高速な NIC を擬似し，開発対象ドライバの処理性能を測定するといった用途が考えられる．

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 ドライバの割込開発手法について述べた．まず既存研究である VM を用いた OS の割込開発手法と問題点について述べた．次に Mint と Mint を用いた割込開発手法について述べた．次に，Mint を用いた NIC ドライバの割込開発環境の方針，課題，および対処について述べた．そして，Mint を用いた割込開発環境の実装について述べた．最後に Mint を用いた割込開発環境の評価について述べた．

Mint を用いた NIC ドライバの割込開発環境では，任意の間隔で割込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，割込間隔と回数の調整，パケットの作成と格納，NIC の状態の更新，割込の発生作成，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，割込管理 AP の作成，開発支援機構の作成，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 ドライバの割込処理を開発対象とした．これを実現する機能として，割込管理 AP，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，割込ハンドラ，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，割込管理 AP は開発支援 OS 上で動作する AP として実装し，割込情報の指定とパケットの作成を行い，開発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，NIC の状態の更新，および IPI の送信については開発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．割込ハンドラの動作，および受信バッファの作

© 1960 Information Processing Society of Japan

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．

実装した開発支援環境を用いて，連続で割込を発生させた際，指定した間隔を守って，割込を発生させられることを示した．また，本環境を用いて実際に NIC ドライバのテストを行った際，NIC ドライバがどの程度の通信速度を実現可能かを調査可能であることを示した．本環境を用いることで NIC ハードウェアを用いずに NIC ドライバの処理性能を調査するといった使用法が考えられる．

参考文献

[1] Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: *An empirical study of operating systems errors*, Vol. 35, No. 5, ACM (2001).
[2] 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).
[3] 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).
[4] 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).
[5] 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).
[6] 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).
[7] Li, J., Si, S., Li, B., Cui, L. and Zheng, J.: LoRe: Supporting Non-deterministic Events Logging and Replay for KVM Virtual Machines, *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, IEEE, pp. 442–449 (2013).
[8] 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).
[9] Renzelmann, M. J., Kadav, A. and Swift, M. M.: SymDrive: Testing Drivers without Devices., *OSDI*, Vol. 1, pp. 4–6 (2012).
[10] Yildirim, E. and Kosar, T.: End-to-end data-flow parallelism for throughput optimization in high-speed networks, *Journal of Grid Computing*, Vol. 10, No. 3, pp. 395–418 (2012).

© 1960 Information Processing Society of Japan