

特 別 研 究 報 告 書

題 目

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

指導教員

報 告 者

藤田 将輝

岡山大学工学部 情報系学科

平成 27 年 2 月 6 日 提出

要約

OS のデバッグ手法として VM を用いたものがある．これは VM 上で 2 つの OS を走行させ，一方の OS から他方の OS へ任意のタイミングで割り込みを発生させることにより，バグを再現し，デバッグを支援するものである．しかし，VM を用いたデバッグ手法では，VM とハイパーバイザ間の処理の遷移に伴う処理負荷が発生する．このため，一定間隔で連続で発生する割り込みや短い間隔で連続で発生する割り込みの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，Mint を用いた OS のデバッグ手法が提案されている．Mint は 1 台の計算機上で複数の OS が論理分割された計算機資源を直接操作できる．また，割り込み処理のデバッグは困難であるが，特に非同期な割り込み処理のデバッグは非常に困難である．この非同期な割り込み処理が頻繁に発生するのが NIC を用いた通信である．本研究では，提案手法を用いて，NIC における割り込みを任意に挿入できる環境を実現する．具体的には，デバッグを支援する OS から NIC ドライバが割り込み処理をするパケットをデバッグを支援する OS とデバッグ対象の OS が共有しているメモリに格納し，割り込み元 OS が占有しているコアから割り込み先の OS が占有しているコアへ IPI を送信することにより，割り込みハンドラが動作し，NIC ドライバが共有メモリからパケットを取得することで割り込み処理を発生させるものである．これにより，NIC ドライバの割り込みにより発生するバグを再現し，デバッグを支援することができる．

目次

1	はじめに	1
2	関連研究	2
2.1	仮想計算機を用いたデバッグ支援機構	2
2.2	概要と処理流れ	2
2.2.1	割り込み挿入法の概要	2
2.2.2	割り込み挿入法の処理流れ	2
2.2.3	ロギング/リプレイ手法の概要	4
2.2.4	ロギングの処理流れ	4
2.2.5	リプレイの処理流れ	5
2.3	問題点	6
3	Mint オペレーティングシステム	8
3.1	Mint の設計方針	8
3.2	Mint の構成	8
3.3	Mint を用いたデバッグ支援環境	9
3.3.1	概要	9
3.3.2	Mint を用いたデバッグ支援環境の処理流れ	11
3.4	Mint における Linux 改変によるバグの影響	11
4	NIC ドライバの割り込みデバッグ環境の設計	12
4.1	目的	12
4.2	NIC ドライバのパケット受信の流れ	12
4.3	NIC ドライバのデバッグ支援環境の構成	14
4.4	設計方針	15
4.5	設計にあたっての課題	16

4.6	課題への対処	17
4.7	対処を実現するために必要な機能	18
5	実装	20
5.1	NIC ドライバのデバッグ支援環境の処理流れ	20
5.2	必要な機能の実現	22
5.2.1	割り込みジェネレータ	22
5.2.2	パケットの作成	22
5.2.3	受信バッファへのパケットの格納	22
5.2.4	受信バッファ状態の更新	22
5.2.5	IPI の送信	23
5.2.6	共有メモリからパケットを取得する割り込みハンドラ	24
5.2.7	受信バッファの作成	24
6	評価	25
6.1	目的	25
6.2	項目	25
6.3	割り込み間隔の評価	26
6.3.1	方法	26
6.3.2	結果と考察	26
6.4	CPU 負荷の評価	26
6.4.1	方法	26
6.4.2	結果と考察	26
7	おわりに	27
	謝辞	28
	参考文献	29

図 目 次

2.1	割り込み挿入法の処理流れ	3
2.2	ロギングの処理流れ	5
2.3	リプレイの処理流れ	6
3.1	Mint の構成	9
3.2	Mint を用いたデバッグ支援環境の処理流れ	10
4.1	NIC ドライバのパケット受信処理流れ	13
4.2	NIC ドライバのデバッグ支援環境の概要	14
5.1	NIC ドライバのデバッグ支援環境の処理流れ	21
5.2	ipi の送信	23

表 目 次

第 1 章

はじめに

近年，OS の多機能化に伴って，OS のデバッグが重要視されている．OS のデバッグ，特に割り込み処理に関するデバッグは非同期的な処理であるため，デバッグが困難になっている．このデバッグを支援する方法として，仮想計算機を用いたものがある．仮想計算機を用いて，デバッグを支援する OS(以下，デバッグ支援 OS) とデバッグ対象の OS(以下，デバッグ対象 OS) の 2 つの OS を動作させ，デバッグ支援 OS 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 OS 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，仮想計算機を用いると，仮想計算機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ため，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 OS である．このため，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同様に存在しなくなる．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．

本論文では，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 Mint のデバッグ支援環境を構築することで，NIC ドライバの割り込み処理のデバッグを支援する環境について述べる．2 章では仮想計算機を用いた既存研究のデバッグ手法の概要，処理流れ，および問題点について述べる．3 章では Mint と Mint を用いたデバッグ支援環境の概要，および処理流れについて述べる．4 章では，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について述べる．5 章では Mint を用いた NIC ドライバの割り込みデバッグ支援環境の実装について述べる．6 章では NIC ドライバの割り込みデバッグ支援環境の評価について述べる．

第 2 章

関連研究

2.1 仮想計算機を用いたデバッグ支援機構

OS のデバッグを支援する環境の既存研究として仮想計算機を用いたものがある．仮想計算機を用いたデバッグ支援環境は大きく分けて 2 つある．割り込み挿入法とロギング/リプレイ手法である．これらについて 2.2 節で説明する．

2.2 概要と処理流れ

2.2.1 割り込み挿入法の概要

割り込み挿入法 [2] はハイパーバイザ上で動作するデバッグ支援 OS とデバッグ対象 OS によって構成される．プログラマがデバッグ対象 OS の割り込みを挿入したいコード位置で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遷移し，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，バグを再現し，デバッグを支援する．

2.2.2 割り込み挿入法の処理流れ

割り込み挿入法ではユーザが割り込みを発生させたいコード位置でハイパーコールを挿入する．割り込みは挿入したコード位置で発生する．また，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する．デバッグ

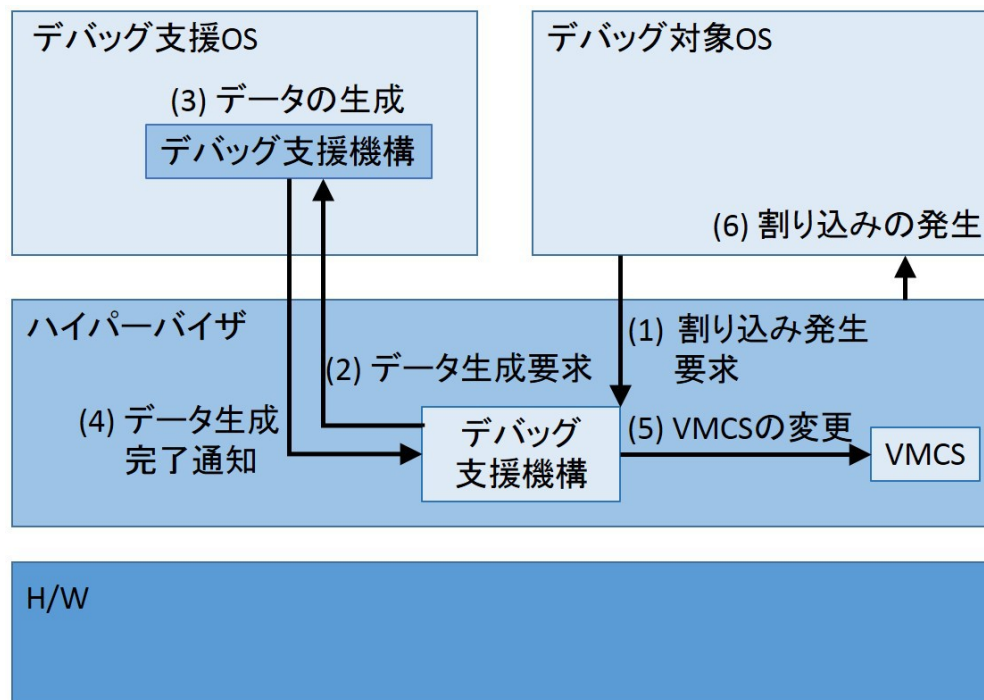


図 2.1 割り込み挿入法の処理流れ

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 2.1 に示し、以下で説明する。

(1) 割り込み発生要求

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。

(2) データ生成要求

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。割り込みに必要なデータとは、パケットやキーコードである。

(3) データの生成

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。

(4) データ生成完了通知

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．

(5) VMCS の変更

ハイパーバイザのデバッグ支援機構が VMCS の内容を変更する．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．

(6) 割り込み発生

デバッグ対象 OS へ処理が遷移し，割り込みが発生する．

2.2.3 ログイング/リプレイ手法の概要

ログイング/リプレイ手法はハイパーバイザまたはハードウェア上で動作するホスト OS と呼ばれる OS 上で動作するデバッグ対象 OS により構成される．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し，再現することで，デバッグを支援する．ここでログイングとは OS の動作の流れを保存することで，リプレイとは保存した流れを再現することである．また，処理の流れを再現するための情報として，以下のような情報がある．これらの情報を以下，再現情報と呼ぶ．

(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

割り込みの種類とこの割り込みが発生したアドレス，および分岐命令を経由した回数である．

(2) 割り込み発生時に使用するデータ

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である．

ログイング/リプレイ手法を用いた関連研究として TTVM[3]，および Sesta[4] がある．TTVM は再現情報に加え，デバッグ対象 OS 側の仮想計算機の状態を保存する．Sesta はログイングを行う 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．

2.2.4 ログイングの処理流れ

ログイングの処理流れについて図 2.2 に示し，以下で説明する．

(1) 割り込みの発生

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．

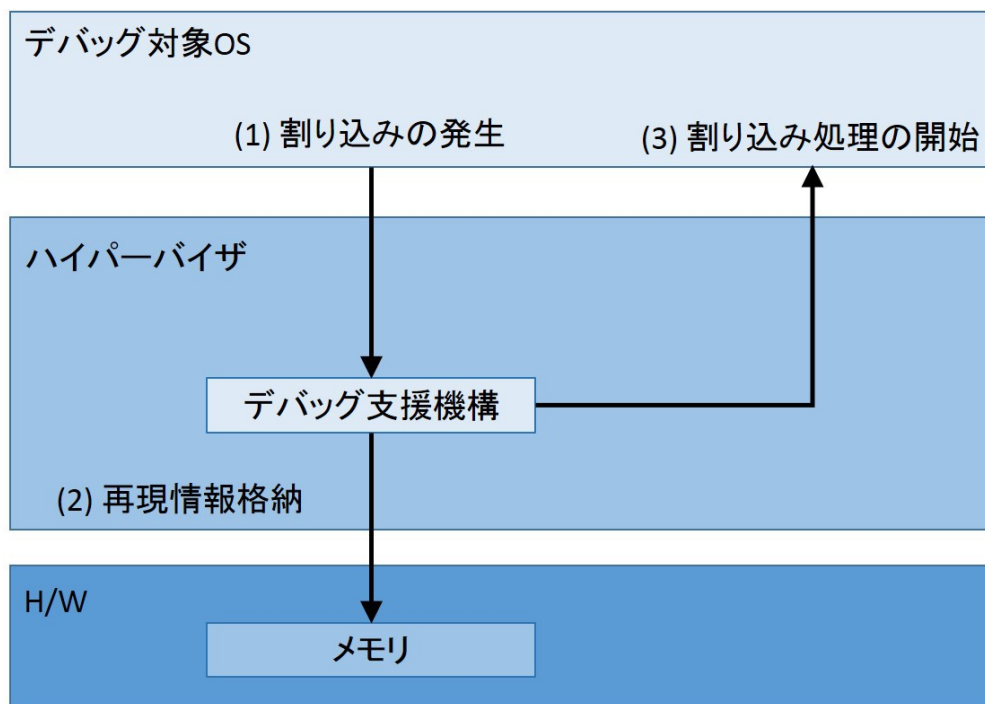


図 2.2 ロギングの処理流れ

(2) 再現情報の格納

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) 割り込み処理の開始

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。

2.2.5 リプレイの処理流れ

リプレイの処理流れについて図 2.3 に示し、以下で説明する。

(1) 再現情報の取得

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) 割り込み発生アドレスまでの処理の実行

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。

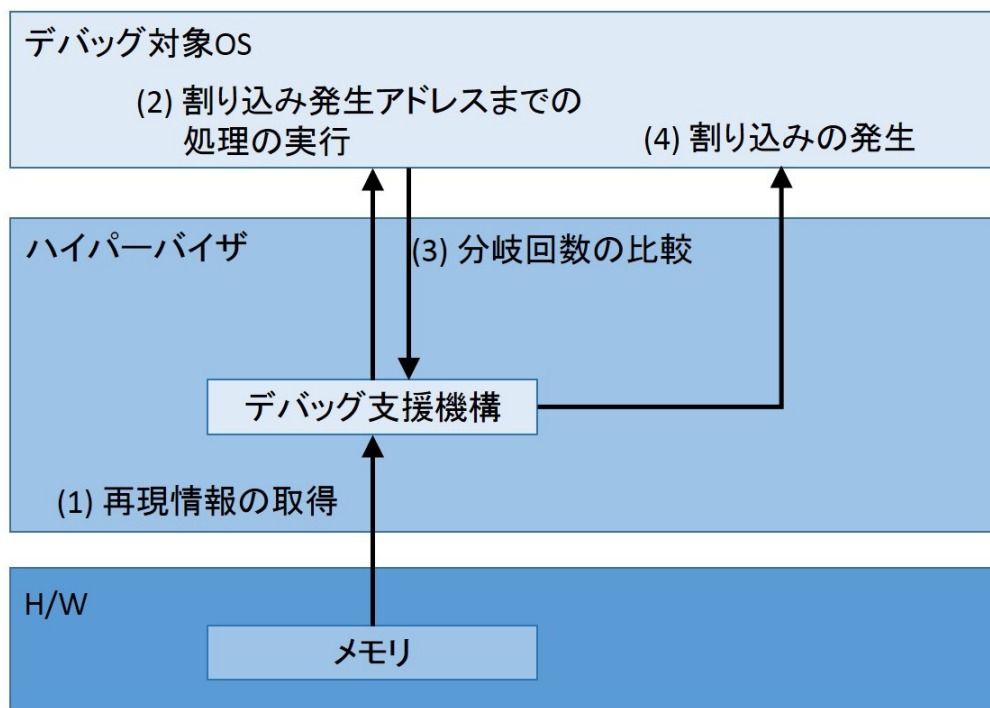


図 2.3 リプレイの処理流れ

(3) 分岐回数の比較

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。

(4) 割り込みの発生

デバッグ対象 OS へ割り込みが発生する。

2.3 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。

(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。コード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、CPU へ発生する間隔で複数の割り込みを発生させようとした際に、この間隔を調整するのはハイパーコールの間隔を調整することで行うが、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．

(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プログラマ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．

(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 とハイパーバイザの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が必要である．

第 3 章

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 CPU、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 3.1 に示し、説明する。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) CPU
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。

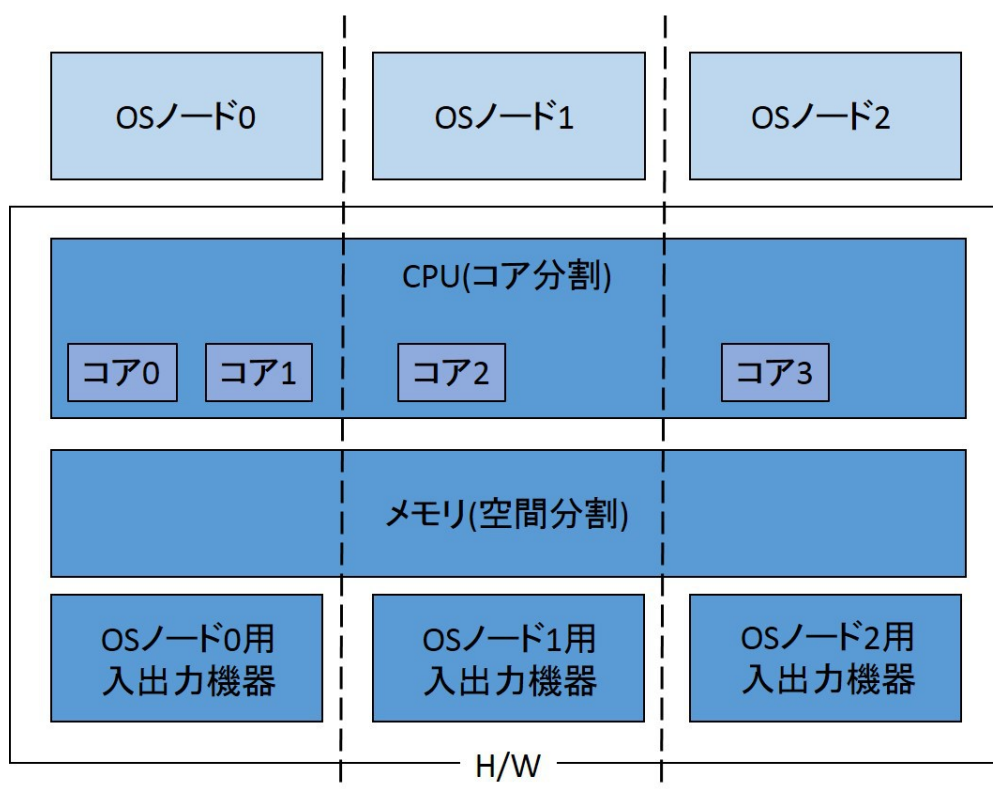


図 3.1 Mint の構成

3.3 Mint を用いたデバッグ支援環境

3.3.1 概要

2章で述べたように、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が必要である。そこで、仮想化を用いずに複数の Linux を動作できる Mint を用いたデバッグ支援環境が提案されている。Mint を用いることで、実割り込みの発生が可能になる。また、任意のタイミングでの割り込み発生も可能になる。

提案されている Mint 用いたデバッグ支援環境の概要について図??に示し、説明する。Mint を用いたデバッグ支援環境は OS ノード 0 としてデバッグ支援 OS が、OS ノード 1 としてデバッグ対象 OS が動作する。デバッグ支援 OS にはデバッグ支援機構があり、デバッグ対象 OS へ割り込みを発生させる。また、割り込みジェネレータをデバッグ支援 OS のアプリケーションとして実装している。これらの割り込みジェネレータ、デバッグ支援機構、デバッグ支援 OS、およびデバッグ対象 OS について以下で説明する。

(1) 割り込みジェネレータ

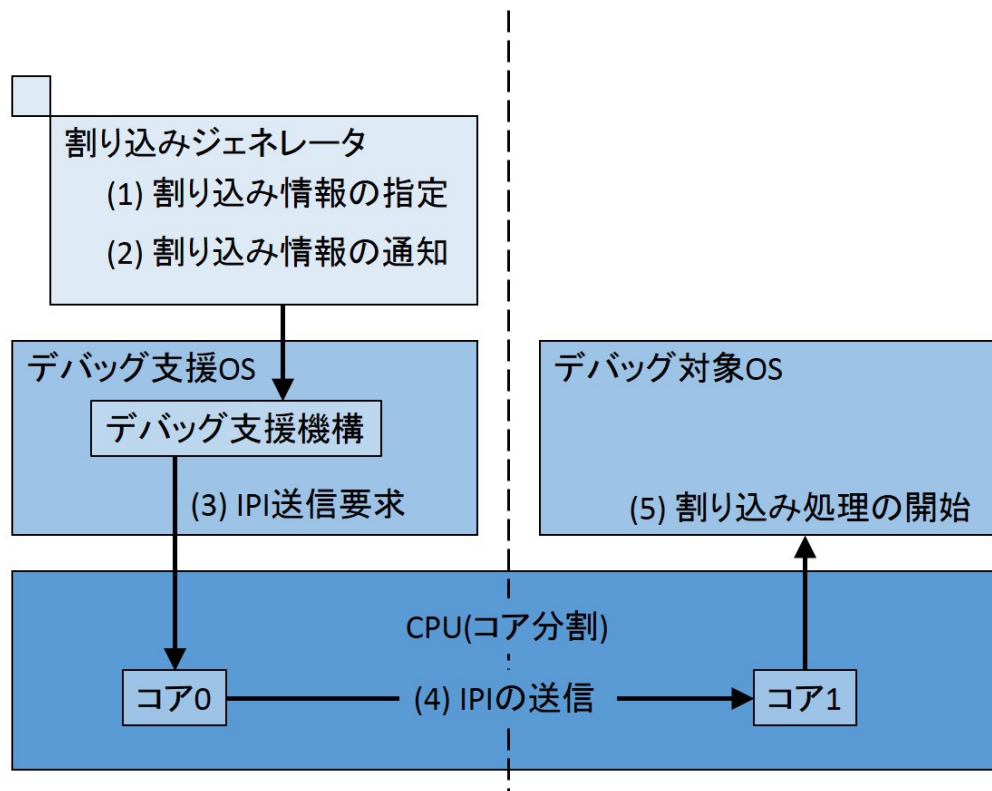


図 3.2 Mint を用いたデバッグ支援環境の処理流れ

プログラマが割り込み情報を指定する際に利用するアプリケーションである。なお，割り込み情報とは割り込みの種類，発生間隔，および発生回数を合わせた情報である。

(2) デバッグ支援機構

割り込みジェネレータから通知される割り込み情報をもとに，デバッグ対象 OS へ Inter-Processor Interrupt(以下，IPI)を送信する機構である。

(3) デバッグ支援 OS

デバッグ支援機構を実装して走行する OS である。ここでは，コア 0 を占有して走行する。

(4) デバッグ対象 OS

デバッグ対象となる OS である。ここでは，コア 1 を占有して走行する。

3.3.2 Mint を用いたデバッグ支援環境の処理流れ

Mint を用いたデバッグ支援環境の処理流れを図 3.2 に示し，以下で説明する．

(1) 割り込み情報の指定

デバッグ支援 OS 上で動作する AP を用いてプログラマが割り込み情報を指定する．

(2) 割り込み情報の通知

割り込みジェネレータがシステムコールを用いて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) IPI の送信要求

デバッグ支援機構がコア 0 へ IPI の送信要求を行う．

(4) IPI の送信

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) 割り込み処理の開始

コア 1 が IPI を受信すると割り込み処理が開始する．

3.4 Mint における Linux 改変によるバグの影響

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 に改変を加える [5]．この際の改変は各 Linux の起動時に認識する CPU，メモリ，およびデバイスを調停するためのものであり，割り込み処理に変更は加えていない．したがって，Mint における Linux 改変におけるバグの影響はないと考えられる．

第 4 章

NIC ドライバの割り込みデバッグ環境の設計

4.1 目的

割り込み処理におけるバグの原因として、非同期な割り込みがある。非同期な割り込みとは、割り込み先と割り込み元でタイミングを合わせないで発生する割り込みのことである。この非同期な割り込みが頻繁に発生するのが NIC ドライバである。NIC ドライバに Mint を用いた割り込みデバッグ支援環境を構築し、非同期な割り込みを再現することで NIC ドライバのデバッグを支援する。また、本研究におけるデバッグ支援環境は NIC ドライバの受信処理に関するデバッグを行うものである。

4.2 NIC ドライバのパケット受信の流れ

NIC がパケットを受信してから NIC ドライバで割り込み処理が発生するまでの流れを図 4.1 に示し、以下で説明する。

(1) 受信バッファアドレスの取得

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。

(2) 受信バッファへのパケットの格納

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。

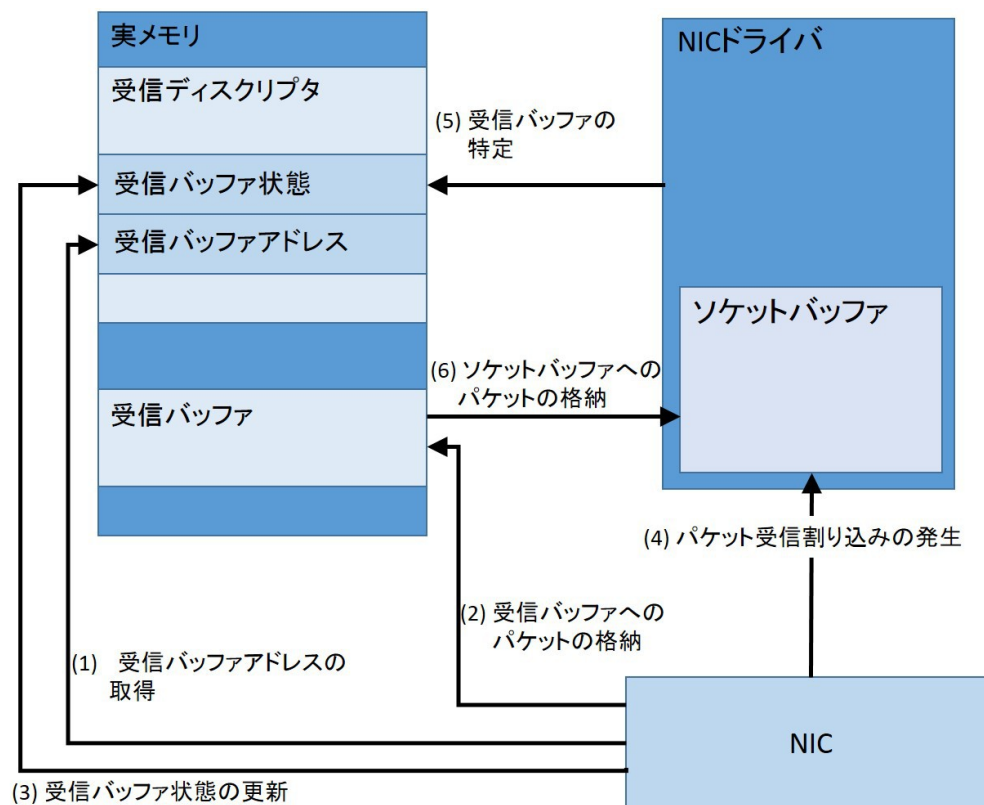


図 4.1 NIC ドライバのパケット受信処理流れ

(3) 受信バッファ状態の更新

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。

(4) パケット受信割り込みの発生

NIC が NIC ドライバに割り込みを発生する。

(5) 受信バッファの特定

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し、受信バッファを特定する。

(6) ソケットバッファへのパケットの格納

NIC ドライバが受信バッファからソケットバッファへパケットを取得し、格納する。

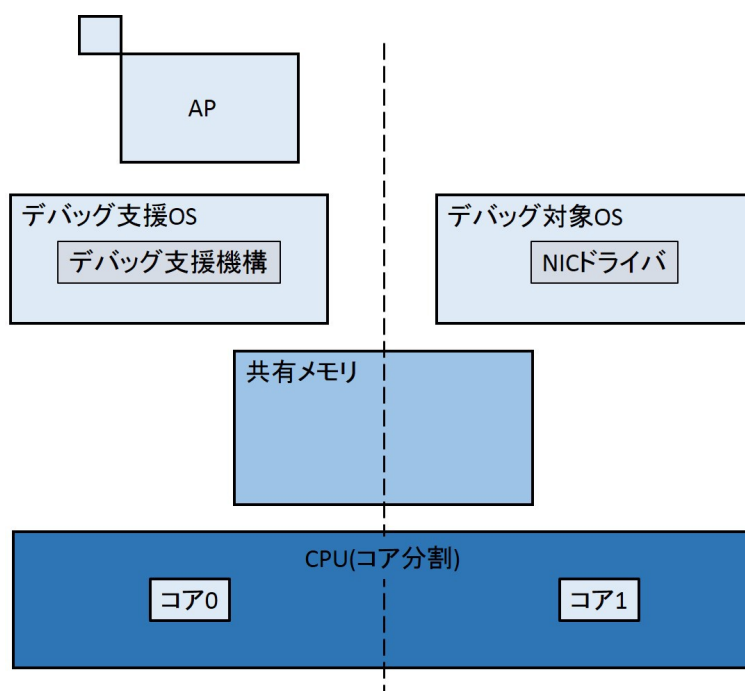


図 4.2 NIC ドライバのデバッグ支援環境の概要

4.3 NIC ドライバのデバッグ支援環境の構成

NIC ドライバのデバッグ支援環境の構成を図 4.2 に示す。NIC ドライバのデバッグ支援環境はデバッグ支援 OS、デバッグ対象 OS、共有メモリ、NIC ドライバ、デバッグ支援機構、CPU、およびアプリケーションによって構成される。それぞれについて以下で説明する。

(1) デバッグ支援 OS

NIC ドライバのデバッグを支援する OS である。この OS は NIC の動作を擬似する。

(2) デバッグ対象 OS

デバッグの対象となる OS である。本研究ではこの OS の NIC ドライバの受信処理のデバッグを対象としている。

(3) 共有メモリ

Mint の共有メモリである。これ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。

(4) NIC ドライバ

デバッグ対象 OS 内で動作している。受信バッファを共有メモリとみなすように改変さ

れている．また，割り込みの契機として IPI を用いる．

(5) デバッグ支援機構

ユーザが指定した情報に基づいてパケットを作成し，共有メモリに配置する．また，コア 0 へ IPI の送信要求を発信する．

(6) CPU

コアを分割してコア 0 をデバッグ支援 OS に，コア 1 をデバッグ対象 OS に占有させている．

(7) アプリケーション

ユーザがパケットの種類，割り込みの発生間隔を指定する．この情報からデバッグ支援機構によりパケットが生成される．

4.4 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機構の設計方針を述べる．

まず，デバッグ支援 OS がデバッグ対象 OS の NIC ドライバへ割り込みを発生させられる環境を構築する．従来の仮想計算機を用いたデバッグ支援環境では，ハイパーバイザへの処理遷移のため実割り込みが困難である．そこで本研究における NIC ドライバの割り込みデバッグ支援環境では実割り込みの再現を可能にする．

次に，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機能を持っている．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，NIC の機能を再現するものが必要になる．そこで NIC の機能を再現するものがデバッグ支援 OS である．デバッグ支援 OS が NIC の機能を再現することで 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．

最後に，NIC の動作をデバッグ支援 OS が再現するため，Mint 上で動作する 2 つの OS 間でパケットの受け渡しが必要になる．これを可能にするため，両 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．

以上の方針を以下にまとめる．

(方針 1) 実割り込みの再現

(方針 2) NIC の動作をデバッグ支援 OS が再現

(方針 3) 共有メモリを用いてのパケットの受け渡し

4.5 設計にあたっての課題

それぞれの設計についての課題について以下に示し，説明する．

課題 1 割り込み間隔，回数の調整

方針 1 の課題として，実割り込みを再現するためには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．割り込みの間隔と回数を指定できる環境が必要．

課題 2 パケットの作成

方針 2 の課題として，NIC ドライバに受け渡すための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 の課題として，NIC の機能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に，デバッグ支援 OS が受信バッファにパケットを格納する機能が必要である．

(課題 4) 受信バッファ状態の更新

方針 2 の課題として，NIC の機能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に，デバッグ支援 OS が受信バッファ状態を書き換える機能が必要である．

(課題 5) 割り込み契機の変更

方針 2 の課題として，NIC の機能の 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，割り込みの契機を変更する必要がある．

(課題 6) 割り込みハンドラの作成

割り込み契機の変更に伴って，NIC ドライバの割り込みハンドラを変更する必要がある．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が必要になる．

(課題 7) 受信バッファの作成

方針 3 の課題として，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．

4.6 課題への対処

課題への対処を以下に示し，説明する．

(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．指定した間隔で指定した回数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 ドライバで処理されるパケットは，パケット中のデータ部分のみである．そこで適当なデータを用意することで NIC ドライバが処理をするパケットとする．

(対処 3) システムコールによるパケットの格納

NIC の受信バッファへのパケットの格納はデバッグ支援 OS のシステムコールにより実現する．対処 2 により作成されたパケットを共有メモリ上に作成した受信バッファに配置する．

(対処 4) システムコールによる受信バッファ状態の変更

受信バッファ状態をデバッグ支援 OS で書き換えるために，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 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なる．

(対処 5) 割り込み契機として IPI を使用

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．IPI は仮想計算機における割り込みのようにハイパーバイザと OS 間の処理の遷移は存在しないため，実割り込みの再現が可能になる．

(対処 6) IPI により動作し，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 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．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，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) 共有メモリへの受信バッファの作成

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に、NIC の受信バッファを共有メモリに作成する必要がある。このために、NIC ドライバの初期化处理内で、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。

4.7 対処を実現するために必要な機能

対処を実現するために必要な機能を以下に示し、説明する。

(機能 1) 割り込みジェネレータ

対処 1 の実現として、デバッグ支援 OS 上で動作する AP を作成し、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。これをシステムコールにより、デバッグ支援機構に通知する。

(機能 2) パケットの作成

対処 2 の実現として、デバッグ支援 OS 内にデバッグ支援機構をシステムコールとして実装し、その処理においてパケットのデータを定義する機能を実装する。

(機能 3) 受信バッファへのパケットの格納

対処 3 の実現として、デバッグ支援機構の処理において機能 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を実装する。

(機能 4) 受信バッファ状態の更新

対処 4 の実現として、デバッグ支援機構の処理において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を実装する。

(機能 5) IPI の送信

対処 5 の実現として、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 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 の送信要求を発行する機能を実装する。

(機能 6) 割り込みハンドラ

対処 6 の実現として、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

ファに格納する機能を持つ割り込みハンドラを実装する。また，NIC ドライバの初期化処理中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 の実現として，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。

第 5 章

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 5.1 に示し、以下で説明する。

(1) 割り込み情報の指定

割り込みジェネレータでユーザが割り込み情報を指定する。

(2) 割り込み情報の通知

デバッグ支援 OS のデバッグ支援機構に指定した割り込み情報をシステムコールにより通知する。

(3) パケットの作成

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) 受信バッファへのパケットの格納

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) 受信バッファ状態の更新

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) 割り込み発生要求

デバッグ支援機構がコア 0 へ IPI 送信要求を行う。

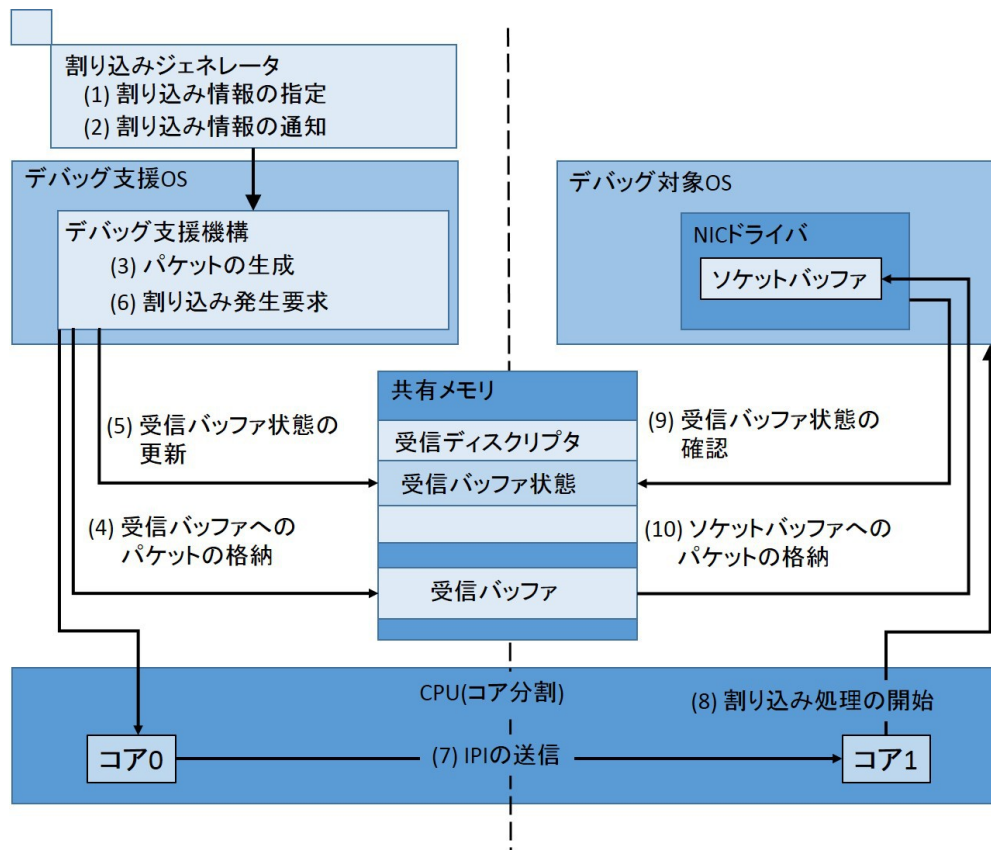


図 5.1 NIC ドライバのデバッグ支援環境の処理流れ

(7) IPI の送信

コア 0 がコア 1 へ IPI を送信する。

(8) 割り込み処理の開始

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) 受信バッファ状態の確認

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) ソケットバッファへのパケットの格納

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

5.2.1 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 として実装する．割り込みジェネレータではパケットを作成す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．これらの情報をシステムコールとして実装するデバッグ支援機構の引数として通知する．

5.2.2 パケットの作成

パケットの作成は，デバッグ支援機構の機能として実装する．NIC ドライバはパケットのデータ部をソケットバッファに格納し，ソケットバッファを上層に送ることで割り込み処理を終了する．したがって，データ部をパケットとして用意できればよい．そこで，適当なデータを定義し，パケットとしている．

5.2.3 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，5.2.2 節で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により定義されたデバッグ支援機構によって実現する．NIC ドライバの受信バッファはリングバッファとなっているため，デバッグ支援機構において擬似的な送信ディスクリプタを用意し，これを用いてリングバッファに対応する．

5.2.4 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，受信バッファにパケットが格納されているか否かの状態 (受信バッファ状態) を持っている．NIC ドライバはパケットの受信処理を行う際，まず受信ディスクリプタ中の受信バッファ状態を確認し，受信済み状態ならば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．そして，こ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．本機構では NIC を使用せず，デバッグ支援 OS が NIC の動作を再現するため，デバッグ

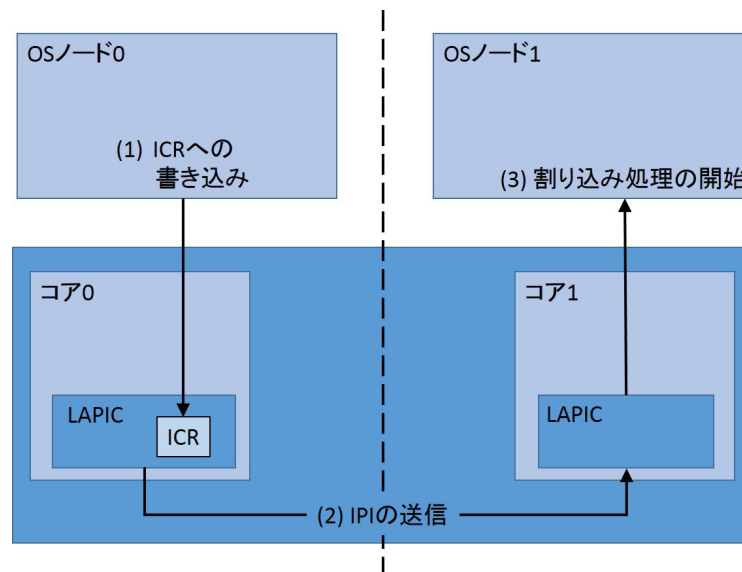


図 5.2 ipi の送信

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。

5.2.5 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に、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割り込み支援機構がデバッグ支援 OS が占有しているコアに IPI の送信要求を発行し、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 5.2 に示し、以下で説明する。

(1) ICR への情報の書き込み

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) IPI の送信

LAPIC ID を参照し、この ID を持つコアへ IPI を送信する。

(3) 割り込みハンドラの動作

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。

5.2.6 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した．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) IPI の受信

デバッグ対象 OS が占有するコア 1 が IPI を受信する．

(2) 割り込みハンドラの動作

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) パケットの取得

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可能にするには，OS 割り込み割り込みハンドラを登録する必要がある．したがって割り込みハンドラの登録ルーチンにより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．これにより，空いている IRQ 番号に割り込みハンドラを登録し，呼び出せるようになる．

5.2.7 受信バッファの作成

第 6 章

評価

6.1 目的

Mint を用いた NIC ドライバの割り込みデバッグ支援環境についての評価を行う．実際の NIC を用いた割り込み処理と本研究のデバッグ支援環境の割り込み処理を比べてどの程度の差があるのかを評価する．これにより，バグが発生した際と同程度の計算機の状態であることを示す．

6.2 項目

評価の項目について以下に示し，説明する．

(1) 割り込み間隔の評価

どれだけ短い間隔で連続の割り込みが発生させられるかを評価する．具体的には，デバッグ支援 OS からデバッグ対象 OS へ間隔を調整して割り込みを発生させ，どの程度の短さまで正常に割り込みが処理されるのかを評価するものである．

(2) CPU 負荷の評価

NIC を使った実際の割り込み処理と比べて，どの程度の CPU 負荷がかかっているのかを評価する．具体的には，NIC を用いた割り込み処理の CPU 負荷を調査し，本研究のデバッグ支援環境を用いた割り込み処理の CPU が実際の割り込み処理の CPU 負荷にどれだけ近付いているかを調査する．

6.3 割り込み間隔の評価

6.3.1 方法

未着手

6.3.2 結果と考察

未着手

6.4 CPU 負荷の評価

6.4.1 方法

未着手

6.4.2 結果と考察

未着手

第 7 章

おわりに

- (1) 本論文の流れを示す .
- (2) Mint を用いたデバッグ支援環境の有用性を示す .
- (3) 本研究では NIC ドライバの割り込み処理に , Mint を用いたデバッグ支援環境を適用させたことを示す .
- (4) 残された課題について述べる .

謝辞

（一例）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] 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., M.C., P.: Debugging operating systems with time-travelling virtual machines, *Proceedings of The USENIX Annual Technical Conference*, pp. 1–15 (2005).
- [4] 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).
- [5] 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).