チェックポインティングとシステムコールエミュレーションを用いたヘテロOS環境のホスト間でのプロセスマイグレーションの実現

中島佳宏[†]　上村佳史[†]
相田祥昭[†]　佐藤三久[†]

今日のグリッド環境上の計算機は，単一の OS 環境から複数の OS が動作するヘテロ OS 環境になることが多く，プログラム作成にはシステムごとの固有のコードを記述する必要がある．また，プログラムの大規模実行では実行時間の長期化が考えられ，その実行中ノード故障やノードの計算からの離脱など，計算環境が動的に変化することがある．そこで我々は，これまでグリッド環境上のヘテロ OS 環境において，プログラムの再コンパイルなしに Linux プログラムを Windows 上で直接実行する方法として，Linux プログラム実行環境 BEE の設計・開発をおこなってきた．さらに，動的な計算環境に対処すべく，我々が開発してきた Linux プログラム実行環境 BEE を拡張し，プロセスのチェックポインティングとリスタートを可能にさせる機能を備えることにより，ヘテロ OS 環境においてプロセスマイグレーションを実現する．本稿では，異なる OS 間でのプロセスマイグレーションを実現する BEE の設計と Linux においてのその実装について述べる．

Design of Process Migration Framework Between Hosts On Which Heterogeneous Operating System Environments Run By Using Process Checkpointing and System Call Emulations

Yoshihiro Nakajima,[†] Yoshifumi Uemura,[†] Yoshiaki Aida[†] and Mitsuhisa Sato[†]

Today's computing environments on a grid are consists of various operating systems such as Windows and Linux. In other words, these computing environments have heterogeneity so that the programmer must write OS-specific code to adapt to each OS. And in case of a large-scale execution of a application, its execution needs long duration furthermore its computing environment changes dynamically because of system failure of nodes or node separation from the computing. Challenges here are to absorb the heterogeneity of operating systems and to adapt for the changing computing environments. We have designed and implemented an agent called BEE, which enables direct execution of Linux binary program on Windows as a prototype to absorb the heterogeneity of operating systems. Moreover extending BEE to add functions to checkpoint a process and to restore the process, we aim to realize process-migration between nodes of heterogeneity OS environment. In this paper, we describe the design of BEE to realize process migration and report its implementation for Linux.

1. はじめに

近年のコモディティPC の普及とネットワークの技術革新により，個人がネットワークに接続された計算機を複数台保有するようになった．しかし，それらのPC は，その計算能力の高さに関わらず，計算能力はほとんど有効に活用されていない．一方で，製薬のための分子構造設計や電子回路設計のシミュレーション，データマイニングなど莫大な量の計算を素早く処理（ハイスループットコンピューティング）するニーズが非常に高く，簡便にこれらの計算資源を含めた大規模計算機環境を利用可能なフレームワークが求められる．

これまでこのハイスループットコンピューティングでは，主な計算資源として Linux サーバからなる PC クラスタを用いて処理が行われており，オフィスや研

† 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻
Computer Science Major, Graduate School of Systems and Information Engineering, University of Tsukuba

究室などで主に利用されている Windows が動作している PC はあまり利用されてこなかった．この Windows が動作している PC の計算能力をハイスループットコンピューティングに活用することができれば，現在ある計算能力の増強または，各個人向け PC の計算能力の有効活用ができるようになる．

そこで我々は，ヘテロな OS 環境において異なる OS 用に作成されたバイナリプログラムを実行可能にさせるミドルウエアとして，Linux 向けに作成されたプログラムを直接 Windows 上で実行可能とする Linux プログラム実行環境 BEE を設計・開発している[1)]．

ハイスループットコンピューティングにおける大規模な計算実行では，計算ノード上で長時間プログラムが実行されることが予想され，その長時間の計算実行中に計算ノードの故障や，計算ノードのその計算からの離脱など，計算環境は動的に変化することが考えられる．

我々は，この動的に変化する環境に向けて，プロセスマイグレーションを用いて対処することを考える．計算ノードの故障の対応については，定期的にプログラムのチェックポイントを行い，最新のプロセスの状態をストレージに保存する．故障した際には，別のノードでそのストレージに保存されたプロセスイメージから再実行をはじめる．また計算からの計算ノードの離脱についても同様に離脱する前にプロセスイメージを保存し，別なホストにイメージを転送して再実行を行う．これによって，途中まで計算した内容が活用可能となり効率的に計算を進めることが可能になる．

また異なる OS 間でプロセスマイグレーションを可能にさせることで，プラットフォームを意識せず計算を行える計算資源をユーザに提供することができるようになる．つまり，異なる OS 環境のホスト間でのプロセスマイグレーションが可能になったことにより，Volatility の高い研究室やオフィスにある個人が所有する PC を，安定した計算資源として活用することが可能になる．さらに，ヘテロ OS 環境でのプロセスマイグレーションが実現可能になれば，これまで申請者が開発してきたグリッド向け RPC システムの一つである OmniRPC[2),3)] で，Windows と Linux それぞれの計算機の計算能力をシームレスに活用することができるようになる．また BOINC[4)] や XtremWeb[5)] などに提案システムを組み込むことができれば，ヘテロ OS 環境の計算機を簡便に活用可能なハイスループットコンピューティングを支援するためのコンピューティンググリッドを実現することが可能になる．

本稿では，この動的に変化する計算環境に対応するため，我々が開発している Windows 上での Linux プログラム実行環境 BEE にプロセスのチェックポインティング機能とリスタート機能を付加させ，Windows と Linux 間でのプロセスマイグレーションの機能を実現するための設計と実装について述べる．

本稿の構成は以下のようになっている．3 章ではこれまで開発してきた Linux プログラム実行環境 BEE の設計と実装について述べる．4 章では，ヘテロ OS 環境におけるホスト間のプロセスマイグレーションのための BEE の設計について説明する．5 章では，Linux における BEE へのプロセスマイグレーションの実装について述べる．最後にまとめと今後の課題について述べる．

2. 関 連 研 究

ハイスループットコンピューティングを支援するミドルウエアにおいて，各 OS 環境に向けてアプリケーションを作成する場合には，プログラマが OS ごとに固有のコードを記述する必要があり，プログラマへの負担は大きい[4)~8)]．

この問題に対処する方法としては，ソースコードレベルでのプログラム動作の互換性を保証する Cygwin ライブラリ[9)] を利用する方法，中間コードを利用して様々な環境でプログラム実行が可能な Java を用いる方法，VMware[10)], QEMU[11)] や Xen[12)] などの仮想マシンを用いて，仮想マシン上に共通の環境を構築することが挙げられる．Cygwin を用いる場合には，ソースコードの再コンパイルが必要となり，Linux に加えて Windows 環境を用意し，Windows のプログラムを用意する必要がある．ハイスループットコンピューティングのためのアプリケーションは，C や FORTRAN で記述されていることが多く，Java で記述されている場合はほとんどないため，実行速度の点からも Java を用いるのは適さない．仮想マシンを用いる場合には，Windows 上に仮想マシンを導入し，さらに Linux の OS を導入しなければならないという問題がある．さらに，仮想マシンは専用のメモリやディスク領域を多く使用するため，OS が仮想マシンに占有されてしまう問題がある．

プロセスマイグレーションについては，これまで単一のシステム環境において行うことはよく行われてきた．しかし，ヘテロ OS 環境でプロセスマイグレーションを行う研究は行われていない．ハイスループットコンピューティングを支援するワークロード管理システム Condor では，プログラムに独自のチェックポインティングライブラリをリンクすることによって，

プロセスが動作する計算機の負荷が高くなった場合に異なる計算機 (UNIX 系 OS が動作) にプロセスマイグレーションを行う[13),14)]．しかし異なる OS ホスト間でのプロセスマイグレーションはできない．VMware ESX server や Xen の仮想マシンを用いた場合では計算のマイグレーション[15)~17)] は比較的容易に実現可能であるが, この処理にはプロセスイメージ以外にも仮想マシンイメージの転送が必要となり, グリッド環境を考慮すると現実的ではない．これまでチェックポインティングとプロセスマイグレーションの機構についての研究は数多く行われてきたが，それらの研究で想定されている OS 環境は単一であることがおおく，異なる OS 環境間での研究は少ない．

3. Linux プログラム実行環境 BEE

我々は，Linux プログラム実行環境 BEE を，ヘテロな OS 環境において Linux から Windows を Grid RPC のワーカとなる計算機資源として利用するためのフレームワークとして設計・開発してきた．本節では Linux プログラム実行環境 BEE について紹介する．

Linux は様々なアーキテクチャに対応しているが，BEE では Linux プログラムを直接 Windows 上で実行することを目的として実装しているため，IA-32 アーキテクチャ上で動作する Linux のプログラムを対象とする．

BEE のシステム概要を**図 1** に示す．BEE のシステムは，2 つの部分から構成されている．1 つはプログラムローダである．Linux では ELF などの多種のプログラムのフォーマットが用いられているが，Windows では PE フォーマットなどがプログラムのフォーマットとなっている．そのため，Linux プログラムのフォーマットは Windows では対応していないため実行することができない．そこで，Linux プログラムのフォーマットを解析，実行するためのプログラムローダの実装をしている．

もう 1 つはシステムコールである．Windows と Linux で同じ IA-32 アーキテクチャを用いた場合，プログラム自体はそのまま実行することは可能である．しかし，システムコールの実装は OS 毎に異なっている．そのため，同じアーキテクチャであっても Windows 上で Linux プログラムをそのまま実行することが出来ない．そこで，Linux のシステムコールを実行するための機能を実装している．

次に，プログラムローダとシステムコールのそれぞれの実装について述べる．

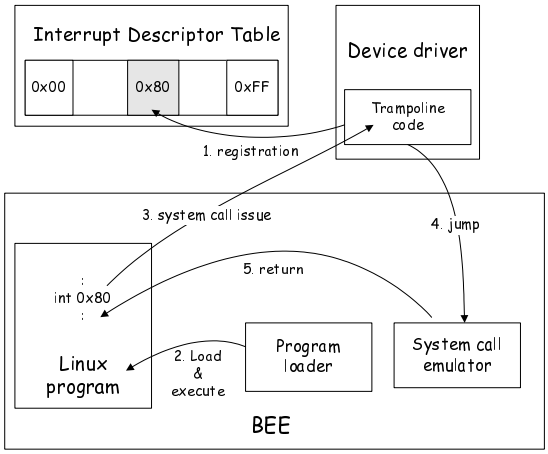

図 1 BEE の構成の概要

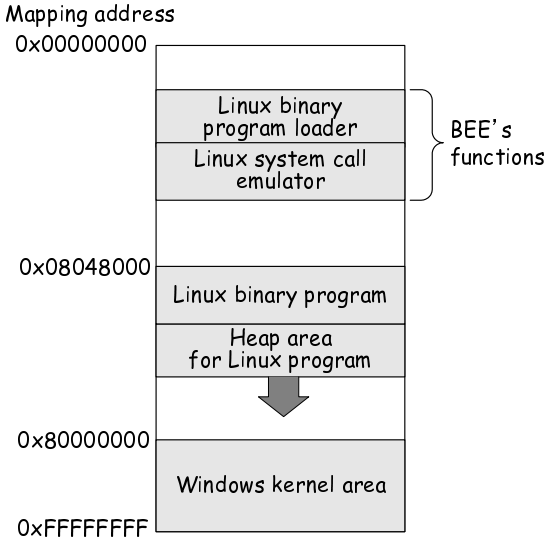

図 2 プログラムローダによる Linux プログラムの展開

3.1 プログラムローダ

Linux のプログラムを実行するには，そのフォーマットに対応したプログラムローダを実装することで実行可能となる．Linux では，プログラムのフォーマットは多くの種類が存在している．BEE では一般的に普及しているフォーマットである ELF[18)] を対象としてプログラムローダの実装を行った．また，ELF には動的リンクと静的リンクがある．BEE ではプログラム単体で実行することが可能となる静的リンクした実行プログラムのみを対象とした．これは，動的リンクの場合，共有ライブラリを Windows が持たなければならなくなるためと，共有ライブラリのバージョンが異なっていた場合に実行することができなくなってし

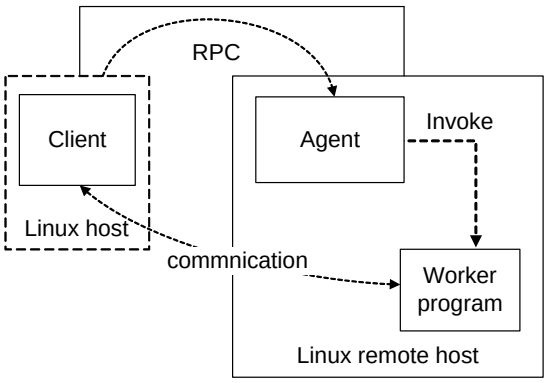

図 3 OmniRPC の RPC システム

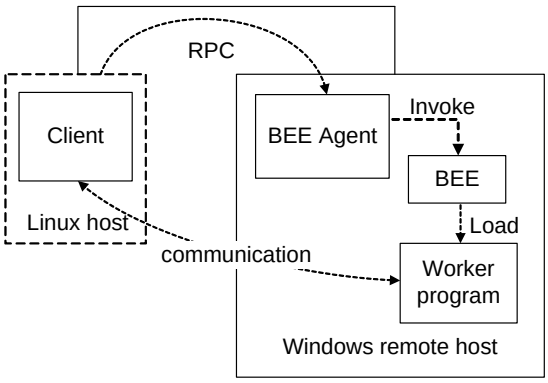

図 4 BEE を用いた OmniRPC の RPC システム

まうという問題が起きるためである．プログラムローダは Linux プログラムのフォーマットの解析を行い，ローダプログラム上にプログラムを展開し実行する．このときの仮想メモリの状態を**図 2** に示す．

3.2 システムコール

入出力などの処理はシステムコールを発行することで実行できる．このシステムコールは OS により異なる実装がなされている．そのため Linux プログラムを Windows 上で実行する場合，システムコールの実装が異なるため実行することができない．Windows でのシステムコールは，NT/2000 の場合は *int 2E*，XP などの場合では *sysenter* 命令を用いて，ソフトウェア割り込みを発生させることで実行している．Linux のシステムコールは *int 0x80* により，ソフトウェア割り込みを発生させ実行している．このように Windows では *int 0x80* に対応する割り込み処理は無いためである．

Windows では Linux のシステムコールに用いられている *int 0x80* に対応する割り込み処理は特に指定されていない．そこで，BEE では *int 0x80* に対応するソフトウェア割り込み処理を，新たに追加することでシステムコールの実行を可能にしている．新たにソフトウェア割り込みの処理を追加する方法として BEE ではデバイスドライバを利用している．

割り込み処理を追加には，割り込みディスクリプタテーブル (Interrupt Descriptor Table)[19] を操作する必要があり，割り込みディスクリプタテーブルに指定する割り込み処理はカーネル上に用意しておかなければならない．通常のユーザプログラムではユーザ空間上にプログラムが展開されてしまうため，これを割り込み処理として割り込みディスクリプタテーブルに指定することができない．しかし，デバイスドライバを用いた場合は可能となる．デバイスドライバは OS の起動時に OS に組み込まれカーネル上に展開される．そのためデバイスドライバを利用して，カーネル上に新たな割り込みの処理を追加することができる．これにより，Linux のシステムコールに対応した処理などの任意のソフトウェア割り込み処理などを実装することができる．デバイスドライバを利用して割り込みディスクリプタテーブルに割り込み処理を追加したときの実行の流れを図 1 に示す．

システムコールのソフトウェア割り込み処理はデバイスドライバを利用することで可能となるが，実際のシステムコールの処理を実装する必要がある．Linux ではシステムコールは多くの種類が存在する．しかし，Windows を Grid RPC のワーカとして利用するのに，このような数多くのシステムコールを実装する必要はない．そこで，BEE ではワーカとして利用するのに必要最低限のシステムコールのみを実装する．現在実装しているシステムコールは write/read などの基本的なファイル I/O と，マスタとのデータのやり取りに必要なネットワーク I/O のみを実装している．

3.3 BEE の OmniRPC への適応

BEE は Linux プログラムを実行する機能しかないため，Grid RPC システムのミドルウェアと統合することで Grid RPC システムに利用することが可能となる．ミドルウェアとして OmniRPC を用いた．ここでは，OmniRPC の概要と BEE を用いた場合の OmniRPC システムについて述べる．

OmniRPC は，ローカルなクラスタ環境から広域ネットワークで構成されたグリッド環境までのシームレスな並列プログラミングを可能にするマスタ/ワーカ型の Grid RPC システムである．OmniRPC のシステムを図 3 に示す．OmniRPC はクライアント，リモートプログラム，エージェントから構成される．クライアントがマスタプログラムである．リモートプロ

グラムはこの図に示されるワーカプログラムであり，ワーカ上で実行するプログラムである．ワーカは，クライアントプログラムからの RPC より呼び出されるが，直接ワーカプログラムの実行は行われない．まず，Linux リモートホストへ ssh などで認証を行った後にエージェントを起動する．そして，クライアントからの RPC はエージェントを通して，ワーカプログラムを呼び出す．その後は，クライアントとワーカプログラムが直接通信を行う．

次に BEE を用いた場合の OmniRPC システムを図 4 に示す．Windows はワーカとしてのみ利用をするため，マスタ側は Linux のみを用いることを前提とする．Windows では ssh などの認証機構がないためエージェントを起動することができない．そこで，Windows でのエージェントをデーモンサービスとして動作するように機能拡張を行った．また，ワーカプログラムは Linux プログラムであるので，エージェントは BEE を経由してワーカプログラムを実行するように変更する．それ以外は Linux での OmniRPC システムと同様に，BEE により実行されたワーカプログラムがクライアントプログラムと通信を行う．

OmniRPC で想定されているグリッド環境は，ネットワーク上で複数のクラスタ環境が接続されている環境である．クラスタ環境では，通常それぞれのクラスタ環境内でファイル共有がされている．しかし，大規模グリッド環境を想定し，オフィスなどの PC を利用する場合では，ファイル共有をすることができない．このような場合にワーカプログラムを各 PC へ配布する機能が必要となる．現在の OmniRPC では，このような機能はサポートされていない．そこで，ワーカ用の Windows PC を簡便に利用できるようにするために，必要なワーカプログラムをマスタが自動的に配布するように拡張した．これにより，Windows PC にワーカプログラムをあらかじめ用意しておく必要がなくなる．

4. 異なる OS 間のプロセスマイグレーションを可能にさせる BEE の設計

我々が対象とするプロセスマイグレーションを行うアプリケーションは，ネットワーク通信やローカルファイルを扱い計算を行うようなプログラムである．ウインドウ表示などの操作を含むようなプログラムは対象としない．**図 5** に，提案するプロセスマイグレーションのフレームワークが使用される想定事例を示す．あるプログラムを実行していたノードにユーザがログインし，負荷が高くなった場合，そのユーザの処理を妨げないようにプロセスのチェックポイントを開始し，スケジューラの指示に従って違うノードにダンプしたプロセスイメージを転送，別なノードで中断したプロセス実行を再開する．ネットワーク通信についてはプロセスマイグレーションのフレームワークとアプリケーション側の協調で対処できるような機構を提供することとする．

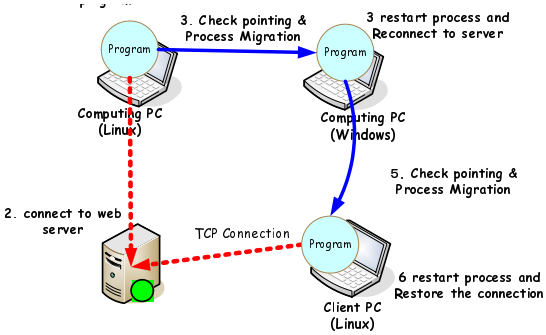

図 5 想定するアプリケーションの実行環境

プロセスマイグレーションを実現する手法は基本的に，OS のカーネルレベルの手法と，ユーザーレベルのみの手法の 2 つがある．我々は，すべてユーザレベルでの手法を選択した．これは，カーネルレベルで行う手法の方がプロセスのメモリや情報などすべてにアクセスできるという利点があるが，Windows のシステム内部についての資料が公開されていないため我々には実現不可能なためである．ユーザレベルではプロセスの完全な情報を取得することは出来ないが，我々が想定するようなアプリケーションではユーザレベルで取得できるプロセス情報で十分であるため問題はないと考える．

異なる OS 間でのプロセスマイグレーションを実現するためには，BEE に以下の機能が必要である．

- プロセスのチェックポイント機能
- チェックポイントイメージからプロセスのリストアを行う機能
- プロセスのリスタート機能
- 複数の OS に対応する共通化したメモリレイアウト
- 仮想ネットワーク I/O の機能

このために，これまで申請者らが開発している，異なる OS 環境向けに作成されたバイナリプログラムを直接実行可能にさせる機構のプロトタイプシステム BEE に，プロセスのチェックポインティング機構とプロセスの状態のリストア機能，プロセスのリスタートの機能を付加させる．

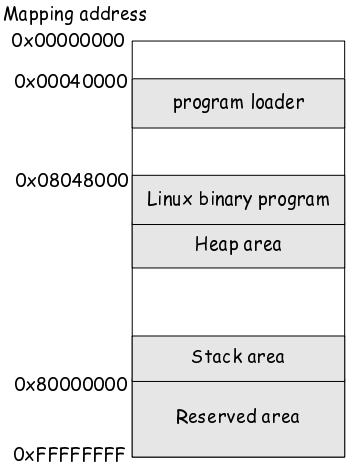

図 6 Windows・Linux 間のメモリマップ

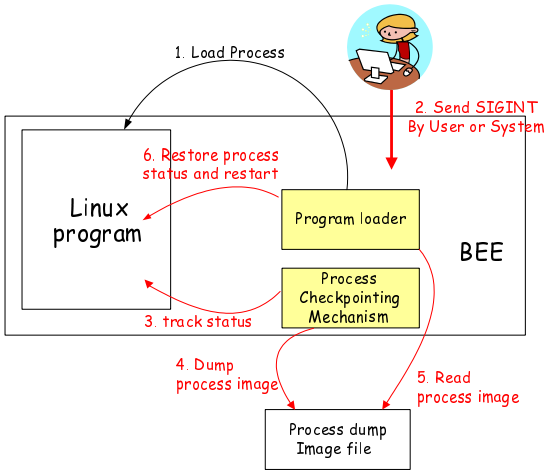

図 7 Linux 環境上のプロセスマイグレーションを可能にする BEE の実装の構成

プロセスのチェックポイントを行う際には，各 OS が提供している API を用いて，他のプロセスから，動作しているプロセスのメモリ情報や CPU のレジスタ情報を読み取り，ELF 形式でプロセスイメージを保存する．

チェックポイントにより生成されるプロセスイメージからのリストア機能とリスタート機能は，これまでの BEE にあるローダを拡張し，プロセスイメージからメモリ空間にプロセスの再構成を行い，各 OS のプログラム実行用の API を用いて実行させる．

また，ヘテロ OS 環境のホスト間において仮想メモリのレイアウトは，各システムによって異なる事が多い．実際，Windows と Linux 間でプロセスマイグレーションを行う場合，スタックなどで使用される仮想メモリのレイアウトは問題となる．WindowsXP を例にとると，デフォルトでは使用可能な仮想メモリの領域は 2GByte である．しかし，Linux の場合は 3GByte が仮想メモリの領域として使用することができる．このように，使用する仮想メモリの領域が異なっていると，プロセスマイグレーションによりプロセスの再構成をする際に，メモリ領域がすでに使用されていて再構成できなくなるという問題が生じる．

そこでプロセスのメモリの使用領域を明確に決定しておく必要がある．BEE では**図 6** のようにメモリ領域を決定する．BEE はこのようにスタックなどの仮想メモリのレイアウト構成するようにし，Windows と Linux 間のプログラムの非互換性を排除するようにプログラムローダの変更を行う．

プロセスマイグレーションにともないプログラムのネットワーク通信は切断されてしまう．このため，ネットワーク通信を必要とするようなアプリケーションのために，BEE を通してネットワーク通信を行うように機構拡張を行う．これによって，ネットワークの再接続などは BEE が対処することができるようになるため，ユーザはプロセスマイグレーションによって生じるネットワーク通信の切断といった問題に対処する必要がなくなる．

5. Linux 環境でのプロセスマイグレーションの BEE への実装

Linux 環境では，プログラムのシステムコールエミュレーションを行う必要がないため，プログラムローダ，プロセスのチェックポイント機能とリスタート機能を実装することによりプロセスマイグレーションを実現することとした．Linux 環境でのプロセスマイグレーションを実現するための BEE の実装を**図 7** に示す．基本的には，プログラムローダでプログラムをメモリ空間に展開，実行を行う．Linux においてのプロセスのチェックポイントを開始するトリガはユーザからのシグナルによって行うこととし，`fork` と `ptrace` を用いてプロセスの情報取得を行う．

チェックポインティング時の処理の流れを図 7 に示す．Linux 上においても BEE と同様にプログラムローダを用いて，Linux プログラムを自身のメモリ空間に展開し実行する．このプログラムローダはこの他にも，先に述べた仮想メモリのレイアウトによる非互換性の排除，ネットワーク通信の再構築といった処理を行う．その後，任意のタイミングのシグナルにより，プログラムローダがシグナルを受け取り `fork` する．`fork` したプロセスに対して `ptrace` を用いてアタッチし，シ

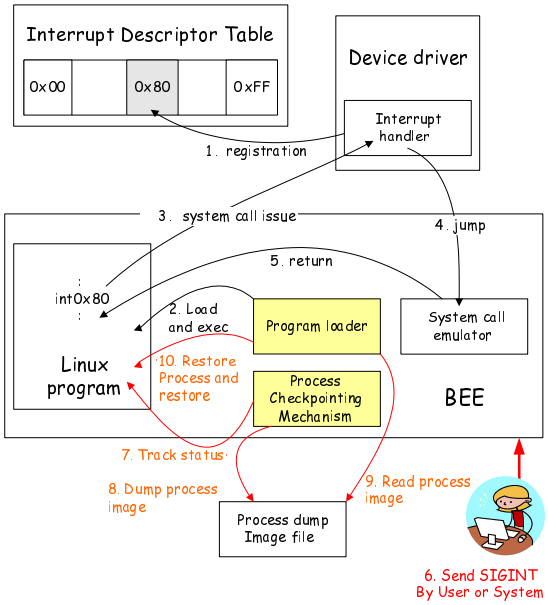

図 8 Windows 環境上のプロセスマイグレーションを可能にする BEE の構成

グナルが発生したときのレジスタやコンテキスト情報を取得する．また，スタックなどのメモリ領域の状態は`/proc/self/maps`から使用されている領域を調べ，プログラムのテキスト以外のデータ領域のみをプロセスイメージとして生成する．プロセスイメージは ELF のコアファイル形式とする．

プロセスのリスタートは，チェックポイント時に生成された ELF コアファイルのプロセスイメージと，実行するプログラムを用いて行う．このコアファイルにはシグナルが発生したときのプロセスの情報とメモリの状態が記録されているため，プログラムローダがプログラムからテキスト，コアファイルからメモリの状態やネットワーク通信などを再構築する．その後，レジスタ情報を書き戻すことでシグナルが発生したときの状態へと復元し，処理を再開する．

6. まとめと今後の課題

本稿では，動的な計算環境に対処するために，プロセスのチェックポインティングとリスタートを可能にさせる機能を付加させ，ヘテロ OS 環境においてプロセスマイグレーションを実現するための Linux プログラム実行環境 BEE の設計と，Linux 環境でのプロセスマイグレーションの BEE への実装について述べた．

今後の課題としては，**図 8** に示すようなプロセスのチェックポインティング機能とリスタート機能を，Windows 環境上の BEE に実装を行い，ヘテロ OS 環境におけるノード間のプロセスマイグレーションの実現を行う．また，マイグレーション機能についてのシステムの基本的な性能評価を行う．

謝辞 本研究の一部は，魅力ある大学院教育イニシアティブ「実践 IT 力を備えた高度情報学人材育成プログラム」，文部科学省科学研究費 補助金 基盤研究 A 課題番号 172002, 特別研究員奨励費 課題番号 17・7324，および，日仏共同研究プログラム（SAKURA）による．

参 考 文 献

1) Uemura, Y., Nakajima, Y. and Sato, M.: Direct Execution of Linux Binary on Windows for Grid RPC workers, *Workshop on Large-Scale and Volatile Desktop Grids (PCGrid) in IPDPS* (to appear).
2) OmniRPC Project: http://www.omni.hpcc.jp/OmniRPC/.
3) Sato, M., Boku, T. and Takahashi, D.: OmniRPC: a Grid RPC ystem for Parallel Programming in Cluster and Grid Environment., *Proceedings of CCGrid2003*, pp. 206– (2003).
4) Berkeley Open Infrastructure for Network Computing: http://boinc.berkeley.edu/.
5) Germain, C., Néri, V., Fedak, G. and Cappello, F.: XtremWeb: Building an Experimental Platform for Global Computing, *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pp. 91–101 (2000).
6) Anderson, D.: BOINC: a system for public-resource computing and storage, *Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 4–10 (2004).
7) Miyazawa, K., Kadooka, Y., Yamashita, T., Suzuki, T. and Tago, Y.: Development of Grid Middleware CyberGRIP and Its Applications, *1st IEEE International Conference on e-Science and Grid Computing, PSE Workshop* (2005).
8) Litzkow, M. J., Livny, M. and Mutka, M. W.: Condor - A Hunter of Idle Workstations., *ICDCS*, pp. 104–111 (1988).
9) Cygwin: *http://sourceware.org/cygwin/*.
10) VMware Inc.: *http://www.vmware.com/*.
11) Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pp. 41–46 (2005).
12) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualiza-

tion, *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ACM Press New York, NY, USA, pp. 164–177 (2003).
13) Litzkow, M., Tannenbaum, T., Basney, J. and Livny, M.: Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, Technical report, Technical Report (1997).
14) Litzkow, M. and Solomon, M.: Supporting Checkpointing and Process Migration Outside the UNIX Kernel, *Proc. of the Winter 1992 USENIX*, pp. 283–290 (1992).
15) Nelson, M., Lim, B. and Hutchins, G.: Fast Transparent Migration for Virtual Machines, *Proceedings of the USENIX 2005 Annual Technical Conference*.
16) Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, pp. 273–286 (2005).
17) ”立薗真樹, 中田秀基, 松岡聡”: ”仮想計算機を用いたグリッド上での MPI 実行環境”, 先進的計算基盤システムシンポジウム SACSIS2006 論文集, pp.525-532, May 2006. (2006).
18) Committee, T.: *Executable and Linking Format (ELF) Specification Version 1.2* (1995). http://www.x86.org/ftp/manuals/tools/elf.pdf.
19) Intel(R): *IA-32 IntelR Architecture Software Developer's Manuals* (2004).