システムソフトウェアと
オペレーティング・システム 89－17
システム評価 2－17
（2002．2．15）

分散環境を指向するプロセス変身機能の提案

石井 陽介 †　　谷口 秀夫 ††

† 九州大学大学院システム情報科学府
†† 九州大学大学院システム情報科学研究院

複数の計算機を結んだ分散環境が広く利用されるようになってきている。しかし、これらの計算機の多くは、元来、UNIX のようなスタンドアロン環境を指向したオペレーティングシステムを利用している。このため、それらのプロセス生成実行機能は、分散環境に適していない。そこで、分散環境を指向したプロセス生成実行機能について述べ、プロセスの変身機能を提案する。プロセス変身機能は、実行プログラムの変更、開始位置の変更、および動作空間の変更という三つの機能から成る。実現例として、***Tender***(The ENduring operating system for Distributed EnviRonment) オペレーティングシステムにおけるプロセス変身機能の実現方式を述べ、プロセス移動処理の基本性能を示す。

Proposal of Process Transformation Facility for Distributed Environment

Yousuke ISHII and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering, Kyushu University

In distributed environment UNIX is used by many computers. But the facility of process creation and execution on UNIX is not proper to distributed environment. Because UNIX is designed for stand-alone environment. In this paper we discuss the facility of process creation and execution for distributed environment and propose process transformation facility. This facility changes the execution environment of a process, such as its program, start point and space. We describe the implementation of the process transformation facility on ***Tender*** operating system. Using it we realize the process migration, and show the performance.

1　はじめに

複数の計算機を結んだ分散環境の利用が進んでいる。この分散環境では、複数の計算機資源を利用できるため、スタンドアロン環境で処理を行うのと比べて、高性能な処理環境を実現することを期待できる。分散環境において、高性能な処理環境を実現するためには、分散環境の特徴を生かした機能を実現する必要がある。

分散環境の特徴を生かした機能の一つの例として、負荷分散がある。これは、システム内の負荷情報を基に、システム内のプロセスを静的、もしくは動的に再配置することである。これより、システム内の計算機資源を有効に利用でき、システムの処理性能を向上させ、高性能な処理環境を実現することが可能になる。分散環境でプロセスの再配置を行う際、静的に行う場合は、プロセスを遠隔計算機上へ生成する機能が必要になる。また、動的に再配置を行う場合は、計算機間のプロセス移動機能が必要になる。つまり、分散環境を指向したプロセスの生成実行機能が必要になる。

しかし、現在のところ、分散環境内の計算機を制御するオペレーティングシステム（以降、OS と略す）の大半は、UNIX のようにスタンドアロン環境を指向した OS である。このため、プロセスの生成実行機能は、分散環境に適したものとは言い難く、負荷分散のような、分散環境を生かした機能をうまく実現できない。

このような背景から、分散環境におけるプロセス生成実行機能については、従来より様々な方法で実現されている。プロセスの遠隔計算機上への生

成法については、次の二つに分けられる。第一に、Amoeba[1] で実現されているように、最初から特定のプロセッサ上に、実行したいプログラムを指定して、新しいプロセスを生成する方法である。第二に、Sprite[1] で実現されているように、UNIX のプロセス生成法と同じ形式を用いる方法である。前者の方法は、処理効率を重視している特徴がある。また、後者の方法は、UNIX アプリケーションとの適合性を重視することで、可用性を高めている特徴がある。計算機間のプロセス移動法については、次の二つに分けられる。第一に、Sprite で実現されているように、OS のカーネルレベルで実現する方法である。第二に、Condor[2] で実現されているように、ユーザレベルで実現する方法である。ユーザレベルで実現する方法の一つとして、UNIX のような既存の OS を利用して、OS のカーネルを変更することなく、ユーザレベルでプロセス移動を実現する方法 [3] もある。前者の方法は、高速処理が実現できる反面、移植性が低いという特徴がある。また、後者の方法は、移植性が高い反面、扱える情報が限られる上に、前者の方法と比べて処理が遅くなるという特徴がある。プロセス移動機能については、文献 [4] にまとめられている。

本稿では、分散環境の特徴を生かし、高性能な処理環境を実現するために、分散環境を指向したプロセス生成実行機能について検討を行った。この中で、プロセス変身機能を提案する。また、プロセス変身機能の実現例として、***Tender*** オペレーティングシステム [5] における実現方式を示し、プロセス移動処理の基本性能を評価する。

2　分散環境におけるプロセス生成実行法

近年、計算機の高性能化と低価格化、および通信路の高速化により、分散環境が広く利用されるようになってきている。現在のところ、この分散環境では、多くの計算機の OS として、UNIX を利用している。

UNIX では、プロセスの生成と実行に fork/exec 方式を用いている。この方式では、fork を発行したプロセス（親プロセス）の環境を、fork によって生成されたプロセス（子プロセス）に引継がせることができる。これにより、入出力の切替え処理が容易に行え、パイプ処理を実現できる長所がある。しかし、fork 時に行う親プロセスの状態複写の内容は、exec 時に変更されるため、CPU 資源とメモリ資源が無駄になる短所がある。これを解決するために、vfork や copy-on-write 機能がある。しかし、vfork は子プロセスが親プロセスのアドレス空間の内容を変更できるので、親プロセスとの整合性が損なわれる恐れがある。また、copy-on-write 機能は制御が複雑な上に、処理のオーバヘッドが大きい。

分散環境では、複数の計算機資源を利用できるため、計算機間の負荷分散や、協調処理による分散処理を実現できる。しかし、元来、UNIX は、スタンドアロン環境での利用を想定して設計された OS である。このため、UNIX で利用されているプロセス生成実行法では、負荷分散のような、分散環境を生かした機能をうまく実現できない。

そこで、分散環境を生かした機能を実現できるように、分散環境におけるプロセス生成実行法について検討を行った。分散環境において、複数の計算機資源を利用することによって、計算機間の負荷分散や、協調処理による分散処理を実現するためには、プロセス生成実行に関し、以下の要求条件を満たす必要がある。

（条件 1）他の計算機上にプロセスを生成できる
（条件 2）他の計算機上へプロセスを移動できる

（条件 1）を実現することにより、負荷分散が実現できる。具体的には、自計算機のメモリ資源や CPU 資源が十分に確保できない場合に、他の計算機上にプロセスを生成し、走行させることが可能になる。また、（条件 2）を実現することにより、以下の三つが可能となる。第一に、プロセスが他の計算機へ移動可能になるため、動的な負荷分散が可能になる。第二に、異なる計算機間で協調処理を行うプロセスの一方を、他方の計算機へ移動可能になるため、プロセス間通信のオーバヘッド削減が可能になる。第三に、保守や点検のために計算機が停止する際、その計算機上に、無停止走行を必要とするプロセスが走行している場合が考えられる。この場合は、プロセスを他の計算機に移動させることで、プロセス停止の回避が可能になる。

さらに、これらを満たした上で、以下の条件がある。

（条件 3）他の計算機上のプロセスと、自計算機上のプロセスとの間でプロセス間通信ができる
（条件 4）他の計算機上のプロセスを削除できる

（条件 3）を実現することにより、異なる計算機上で走行するプロセス間で協調処理を行えるようになるので、システム内で分散処理が実現可能になる。ま

た、（条件 4）を実現することにより、他の計算機上に生成したり、移動させたプロセスが正常に動作しなくなった場合や、走行する必要がなくなった場合には、そのプロセスの削除が可能になる。

ここでは、プロセスの生成実行の観点から、（条件 1）と（条件 2）に着目して考える。

（条件 1）を満たすには、プロセス生成先の計算機を陽に指定でき、さらに、処理を高速かつ省資源で行える必要がある。なぜなら、負荷分散を効率的に行うには、プロセスを任意に指定した計算機に生成でき、かつ負荷分散の効果を低下させないように低い負荷で負荷分散処理を行う必要があるためである。ここで、分散環境におけるプロセス生成実行法として、fork/exec 方式を用いると、fork 時に行う親プロセスの状態複写時に利用する CPU 資源とメモリ資源が無駄になり、好ましくない。このため、プロセス生成実行法には、プロセスが利用するためのアドレス空間を生成した後に、直接プログラムをロードする方式が良い。例えば、形式 create_process(program_id,processor_id) で、program_id で実行プログラムを、processor_id で動作するプロセッサを指定できるようにする。これにより、プロセス生成時の無駄なメモリ間複写が省け、プロセス生成処理の高速化、および省資源化が期待できる。

また、（条件 2）を満たす際、計算機間のプロセス移動を他の関連する機能と統合して行えれば、高度なサービス実現が可能になる。そこで、プロセスの構成要素を変更し、プロセスの実行環境を変更する機能として、プロセス変身機能を提案する。以降では、プロセスの変身機能について詳しく述べる。

3　プロセス変身機能

3.1　プロセスの構成要素

プロセスとは、プログラムを実行する際、OS がその動作を制御する基本単位である。プロセスは、様々な資源から構成されている。プロセスの構成要素を図 1 に示す。プロセスの構成要素には、プログラム、プロセス管理表、プログラムの実行のために必要なもの（以降、内部資源と呼ぶ）、プログラムの処理が必要とするもの（以降、外部資源と呼ぶ）がある。プログラムは、テキスト部、データ部、BSS 部、スタック部（ユーザスタック部、カーネルスタック部）からなる。テキスト部は、プロセッサが実行可能な命令の列である。データ部は、初期値を持つ変数や文字列の集合部分である。BSS 部は、初期値

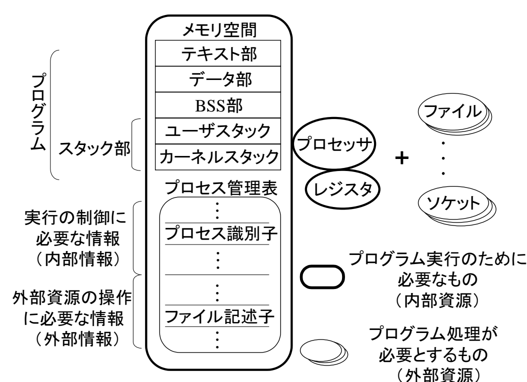

図 1　プロセスの構成要素

を持たない変数の集合部分である。スタック部には、ユーザスタックとカーネルスタックがあり、プロセスがそれぞれ、ユーザモードまたはカーネルモードで走行する時に利用する。プロセス管理表が持つ情報は、実行の制御に必要な情報（以降、内部情報と呼ぶ）とプログラム処理が必要とする外部資源の操作に必要な情報（以降、外部情報と呼ぶ）に分類できる。内部資源には、仮想記憶空間、プロセッサ、レジスタ群等があり、外部資源には、ファイル、ソケット等がある。

3.2　プロセス変身を実現する機能

3.2.1　基本方針

プロセス変身機能とは、プロセスの構成要素を変更し、プロセスの実行環境を変更する機能のことである。ここでは、プロセスの構成要素の内、プロセス実行時に必須であるプログラムとメモリ空間について着目した。これらを当該プロセスの実行途中に変更することで、プロセスの変身を実現することを目指した。

ここで、プログラムを変更する際には、次の三つの場合が考えられる。

（1）別のプログラムに変更して、そのプログラムの先頭から処理を行う

（2）プログラムは変更せず、そのプログラムの開始位置を変更し処理を行う

（3）別のプログラムに変更して、そのプログラムの開始位置を変更し処理を行う

これらの中で、場合（1）と場合（2）に対応できる機能を独立に実現すれば、それらの機能を組み合わせて利用することにより、場合（3）についても対処可能になる。このため、プログラムを変更する際には、場合（1）と場合（2）に対応する機能をそれ

ぞれ実現することにした。

また、メモリ空間を変更する際には、プロセスの動作空間、すなわち、プロセスが走行する際に利用する仮想記憶空間に着目した。ここでは、動作空間の変更を、変更前と変更後の仮想記憶空間が存在する計算機が同一であるか否かを意識せず、位置透過に実現することにした。このため、動作空間の変更により、プロセスが利用する計算機、およびプロセッサも変更できるようになる。

以上より、プロセス変身のための機能として、次の三つの機能を実現することにした。

（機能 1）実行プログラムの変更

（機能 2）開始位置の変更

（機能 3）動作空間の変更

以降に、これら三つの機能の詳細について述べる。

3.2.2 実行プログラムの変更

実行プログラムの変更機能とは、プロセスとして実行されるプログラムを別のプログラムに変更する機能である。実行プログラムを変更したプロセスは、変更後のプログラムの先頭から処理を行うことになる。これにより、複数のプログラムを一つのプロセスとして逐次的に実行できる。

3.2.3 開始位置の変更

開始位置の変更機能とは、変更対象プロセスが次に走行する開始位置を変更する機能である。開始位置の変更には、次の二つの機能がある。

（1）初期状態に変更

（2）指定された位置に変更

機能（1）により、プロセスの再起動 [6] が可能になり、プロセス生成と消滅のオーバヘッドを削減できる。機能（2）により、チェックポイント機能を利用してプロセスの任意の位置における状態を保存しておき、プロセスを保存した状態から再開させることができる。これにより、プロセスの実行途中で起る障害からの高速な回復処理を実現できる。

3.2.4 動作空間の変更

動作空間の変更機能とは、プロセスが利用する仮想記憶空間を別の空間に変更する機能である。動作空間の変更には、次の二つの機能がある。

（1）同一計算機内の別空間に変更

（2）他計算機上の空間に変更

機能（1）により、プロセス間の処理の疎密度に合わせた空間の変更や、データが存在する空間へプロセスが移動することによるデータの排他処理が可能になる。機能（2）により、計算機間のプロセス移動が実現できる。このため、負荷分散、協調処理のオーバヘッド削減、および保守や点検による計算機の停止に伴うプロセス停止の回避が可能になる。

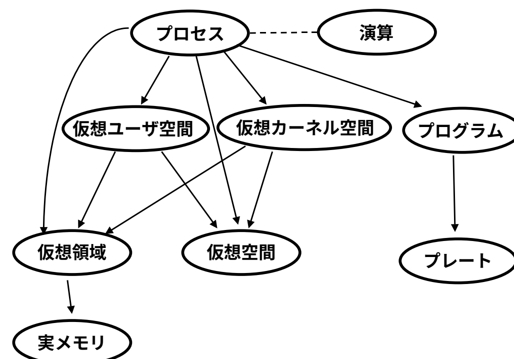

図 2 プロセスを構成する資源

3.2.5 複数の機能を組み合わせた利用

上記の三つの機能を利用する形式は、それぞれ単独で利用する形式だけでなく、複数の機能を組み合わせて利用する形式も可能である。これにより、複数機能を利用して処理を行う際に、その処理の呼び出し回数を削減できる。特に、処理要求が計算機間にまたがって行われる場合には、処理に必要となる通信回数を削減できる。以上より、複数の機能を組み合わせて利用できるようにすることは、処理の高速化を行うために有効である。

4 実現例

プロセス変身機能の実現例として、***Tender*** オペレーティングシステムに本機能を実現した。

4.1 *Tender*

Tender は、プログラム構造を重視し、OS の操作対象を資源として分離し独立化させている。ここで、***Tender*** のプロセスを構成する資源を図 2 に示す。矢印は、処理の依存関係を表している。プロセスの構成資源には、資源「プログラム」と資源「プレート」がある。資源「プログラム」とは、プログラムのテキスト／データのサイズと先頭アドレス、および、プログラムの開始アドレスの情報からなり、プログラムの実行形式を隠蔽している。プログラムの内容は、プレート上に存在している。ここで、資源「プレート」とは、永続的な記憶を提供するものであり、既存の OS のファイルに相当する。また、資源「演算」は、プロセスへのプロセッサ割り当て単位を資源化したもので、プロセスとは独立して存

在する。プロセスは、演算を確保することで、プロセッサの割り当てを受けて走行できる。

また、***Tender*** では、ヘテロ仮想記憶 [7] を実現している。ヘテロ仮想記憶とは、単一仮想記憶と多重仮想記憶を融合した仮想記憶モデルである。ヘテロ仮想記憶では、複数の仮想記憶空間を提供し、一つの仮想記憶空間内に 0 個以上のプロセスが存在でき、プロセスが仮想記憶空間の間を移動できる。

4.2　特徴

Tender でプロセス変身機能を実現する際、次の点を考慮することによって、処理の高速化を図ることにした。

（1）資源の事前用意や保留を行う

（2）ユーザプロセスのみを適用対象としてカーネルスタックを扱わない

まず、項目（1）について述べる。***Tender*** では、資源の分離独立化により、資源の事前用意や保留が可能になっている。これにより、資源の生成や削除を伴う処理を高速化することができるため、処理の高速化を図ることが可能となる。具体的には、プロセス変身処理において、変身対象のプロセスが変身前に利用していた資源を、変身後に削除するのではなく、保留しておくことができる。これにより、資源の削除処理時間の高速化が図れる。さらに、保留していた資源を、当該の資源生成時に再利用することにより、資源の生成処理時間の高速化も図れる。

次に、項目（2）について述べる。プロセスは、カーネルプロセスとユーザプロセスの二つに分類することができる。前者は、生成されてから消滅するまでカーネルモードで走行する。このプロセスは、主に、OS の機能を提供するために利用される。一方、後者は、主にユーザモードで走行する。ただ、システムコールを発行することによって OS の機能を利用する時には、カーネルモードで走行する。このプロセスは、主に、ユーザによって生成され、ユーザがプログラム処理を行うために利用される。ここで、プロセス変身機能の適用対象とするプロセスについて考える。本機能により実現できるのは、プロセス再起動によるプロセス生成実行の高速化、およびプロセス移動による負荷分散等である。これらの機能をユーザに提供するためには、プロセス変身機能をユーザプロセスに対して適用できるようにすればよい。このため、***Tender*** で実現するプロセス変身機能は、その適用対象をユーザプロセスとすることにした。また、変身対象となるプロセスの構成要素には、プログラムのテキスト部、データ部、およびユーザスタック部のように、ユーザモードで走行する時に利用するものと、カーネルスタックのようにカーネルモードで走行する時に利用するものがある。ここで、ユーザに対して、プロセス変身機能を利用したサービスを提供する場合は、前者のユーザモードで走行する時に利用するもののみを対象として扱うことにより、必要なサービスを提供できる。このため、プロセス変身処理においては、ユーザモードで走行する時に利用するもののみを、処理の対象として扱うことにした。また、変身後のプロセスは、変身前に利用していたカーネルスタックの内容に関係なく、指定されたプログラムの開始位置から処理を開始できるようにした。これにより、カーネルスタックを扱う必要がなくなり、変身処理の高速化を図ることができる。以上のことを実現するために、プロセス変身処理を行う際、当該プロセスのカーネルスタックに、プログラムの開始位置となるアドレスを格納することにした。これにより、当該プロセスが次にディスパッチされた時に、格納されているアドレスを基にして、変身後の処理を開始できるようにした。

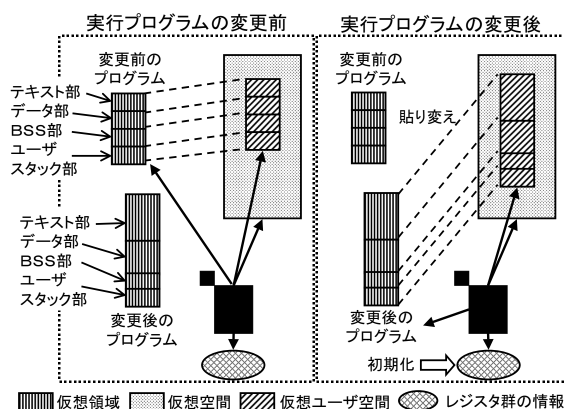

図 3　実行プログラムの変更

4.3　実現方式

4.3.1　実行プログラムの変更

実行プログラムの変更の様子を図 3 に示す。図において、仮想領域は、実メモリあるいは外部記憶装置のデータ格納域情報を仮想化した資源である。仮想空間は、仮想アドレスの空間であり、仮想アドレスを実アドレスに変換する変換表に相当する。仮想ユーザ空間は、プロセッサが仮想アドレスによって

アクセス可能な空間であり、仮想領域を仮想空間に「貼り付ける」ことで生成され、「剥す」ことで削除される。ここで、「貼り付ける」とは、仮想空間が持つアドレス変換表に、当該の仮想領域のデータ格納域情報を設定することに相当する。逆に、「剥す」とは、貼り付けた際に設定したデータ格納域情報を解放することに相当する。

実行プログラムの変更時には、以下の処理を行う。変更後のプログラムが既に仮想領域にロードされている場合は、プロセスが利用している仮想領域を変更後のプログラムがロードされているものに変更し、仮想空間への貼り変えを行う。ここでは、変更前に利用していた仮想領域を仮想空間から剥し、変更後に利用する仮想領域の貼り付けを行うことで貼り変えを行う。ロードされていない場合は、新規に仮想領域を生成し、仮想空間への貼り変えを行ってから変更後のプログラムをロードする。

4.3.2 開始位置の変更

開始処理の変更の様子を図 4 に示す。初期状態に変更する際は、データ部、BSS 部、ユーザスタック部、およびレジスタ群の内容を初期化する。初期化時には、当該プロセスが利用している資源「プログラム」と、プログラムの内容が存在する資源「プレート」を利用する。資源「プログラム」からはプログラムの開始アドレスなどのプログラムについての情報を獲得する。資源「プレート」は、データ部と BSS 部の初期化時に利用する。

指定された位置に変更する際は、当該の指定された位置における情報が必要となる。この情報は、プロセスが利用するデータ部、BSS 部、およびユーザスタック部の内容と、レジスタ群の情報からなる。ここでは、指定された位置における情報をデータ部、BSS 部、ユーザスタック部、およびレジスタ群へ複写する。これにより、このプロセスが次にディスパッチされた時に、指定された位置から走行を開始できる。

4.3.3 動作空間の変更

動作空間の変更の様子を図 5 に示す。同一計算機内の別空間に変更する場合は、仮想領域の貼り変えを行う。ここでは、変更元の仮想空間から当該の仮想領域を剥し、その仮想領域を変更先の仮想空間へ貼り付けることで貼り変えを行う。

また、他計算機上の空間に変更する場合は、まず始めに、変更先の計算機の仮想空間上にプロセスを生成する。次に、生成したプロセスの実行プログラ

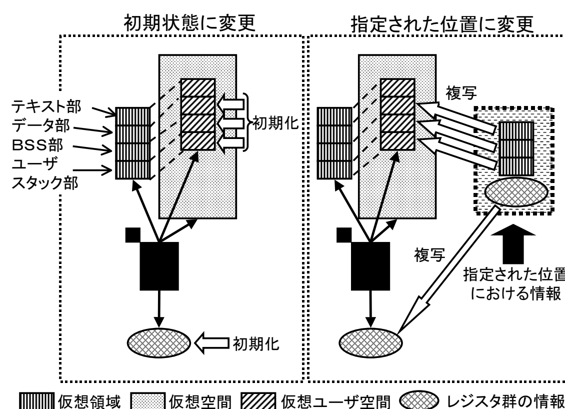

図 4 開始位置の変更

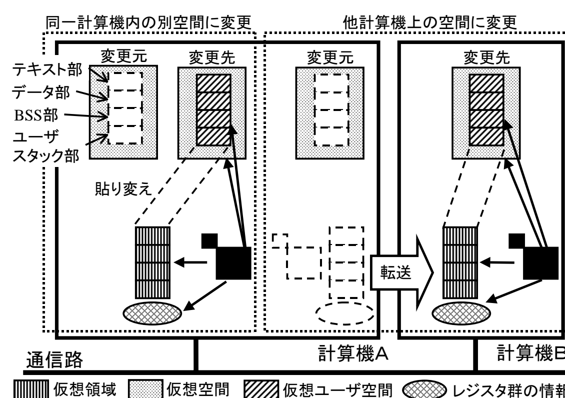

図 5 動作空間の変更

ムと開始位置を、プロセスの変身機能を利用して、動作空間を変更するプロセスのものへ変更する。このとき、実行プログラムと開始位置の変更処理に必要な情報を、処理要求と同時に変更先計算機へ転送する。変更先計算機では、その転送された情報を利用して、当該プロセスの実行プログラムと開始位置の変更を行う。その後、最後に、変更元計算機上のプロセスの削除を行う。

4.3.4 複数の機能を組み合わせた処理

プロセス変身機能は、上記の三つの機能を単独で利用する形式だけでなく、各機能を組み合わせて利用する形式も実現した。その際、次の二つの場合は、処理の組合わせによる無駄な処理を省くことにより、処理の高速化を行うことが可能である。

（1）実行プログラムと開始位置の変更を組み合わせて行う場合

（2）他の計算機上への動作空間の変更するとき、実行プログラムや開始位置の変更を組み合わせて行う場合

表 1　提供する基本インタフェース

変更対象	形式	機能
実行プログラム	trans_proc(pid,plateid,argv)	pid で指すプロセスのプログラムを plateid で指すプログラムに変更し、引数 argv を渡す。
開始位置	restart_proc(pid,restart_arg,argv)	pid で指すプロセスの開始位置を restart_arg で指定された位置に変更する。もしくは、初期状態に変更して引数 argv を渡す。
動作空間	move_proc(pid,vmid)	pid で指すプロセスの動作空間を vmid で指す空間に変更する。

表 2　統合インタフェース

形式	説明
reset_proc(pid,proc_op,plateid,restart_arg,vmid,argv)	pid で指すプロセスを変身させる。処理内容は proc_op により以下のように指定する。 proc_op == 1：実行プログラムの変更 proc_op == 2：開始位置の変更 proc_op == 4：動作空間の変更 (論理和を用いた複数の機能の指定も可能) 残りの引数は必要な分だけを設定する。

ここでは、これらの場合について対処を行い、処理の高速化を行った。以降に、各場合における対処について述べる。

場合（1）は、二つの変更処理が、当該プロセスのデータ部、BSS 部、ユーザスタック部、およびレジスタ群に対して、中身のデータの読み込み処理を行うため、対象となる領域への処理が重なってしまう。このため、まず、場合（1）における処理の手順は、実行プログラムの変更をしてから開始位置の変更を行うことにした。さらに、実行プログラムの変更時に仮想領域などの新規に必要となる資源の生成だけを行い、中身のデータの読み込みは行わないようにすることにした。中身のデータについては、開始位置の変更時に読み込みを行うようにした。これにより、データの読み込み処理が一度で済むようになる。

場合（2）は、他の計算機上へ動作空間を変更する際、あらかじめ変更先計算機に生成したプロセスを、動作空間を変更させるプロセスの実行プログラムと開始位置に変更している。したがって、動作空間を変更する処理の前、もしくは後に、組み合わせて行う実行プログラムや開始位置の変更処理を行っても、場合（1）と同じように、対象となる領域への処理が重なってしまう。このため、場合（2）における処理は、動作空間の変更処理中において、変更先計算機に生成したプロセスの実行プログラムと開始位置を変更する時点で、組み合わせて行う実行プログラムや開始位置の変更を行うことにした。これにより、動作空間の変更処理時間のみで、動作空間の変更と組み合わせて行う実行プログラムや開始位置の変更処理も、一緒に行えるようになる。

4.3.5　提供インタフェース

プロセス変身機能は、表 1 に示す基本インタフェースにより利用できる。また、表 1 に示す三つのインタフェースを統合したインタフェースを作成した。この統合インタフェースを表 2 に示す。これにより、一回の呼び出しで、複数の機能を組み合わせた処理を行うことができる。

4.4　評価と考察

プロセス変身機能を利用したプロセス移動処理について、基本性能を評価した。

測定は、Myrinet[8] により接続された Pentium II 450MHz の計算機二台を用いて行った。Myrinet とは、1.28Gbps の通信性能を持つ通信路のことである。

測定した項目は次の二つである。同じ処理を 50 回繰り返して行い、その平均時間を求めた。

（1）同一計算機内で新たに仮想空間を生成し、生成した仮想空間にプロセスを移動させた場合
（2）遠隔の計算機上に新たに仮想空間を生成し、生成した仮想空間にプロセスを移動させた場合

また、移動させるプロセスの BSS 部とユーザスタック部のサイズを 4KB に固定し、テキスト部とデータ部のサイズを変化させ、移動時間との関係を調べた。測定した移動時間とテキスト部サイズとの関係を図 6 に示し、データ部サイズとの関係を図 7 に示す。

図 6 は、移動プロセスのデータ部サイズを 4KB

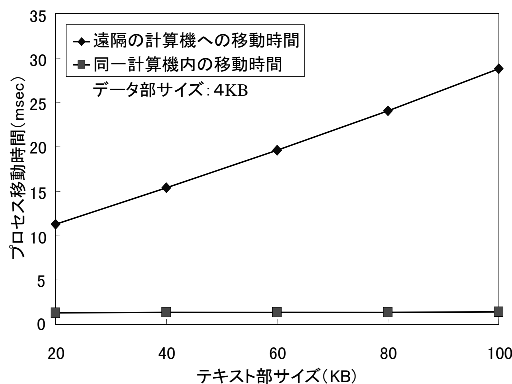

図 6 プロセス移動時間とテキスト部サイズとの関係

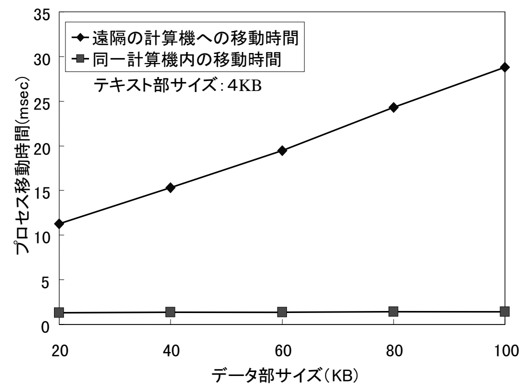

図 7 プロセス移動時間とデータ部サイズとの関係

に固定して、テキスト部サイズを変化させた場合のプロセス移動時間を表したグラフである。これより、同一計算機内のプロセス移動時間は、テキスト部サイズに比例して増加するものの、その増加の割合は非常に小さくほぼ一定であることがわかる。一方、遠隔の計算機へのプロセス移動時間は、同一計算機内の移動時間と比べて、テキスト部サイズの増加に比例して大きく増加している。テキスト部サイズの増加に比例して移動時間が増加する原因としては、計算機間のデータ転送時間、およびメモリ間の複写処理時間が考えられる。

図7は、移動プロセスのテキスト部サイズを4KBに固定して、データ部サイズを変化させた場合のプロセス移動時間を表したグラフである。図6と図7を比較すると、同一計算機内の移動時間、および遠隔の計算機への移動時間は、同様な値となっている。

以上より、同一計算機内の移動処理と遠隔の計算機への移動処理のどちらとも、プロセス移動時間は、テキスト部、または、データ部という種類は関係なく、それらの大きさが影響していることがわかる。

5 おわりに

分散環境を指向したプロセス生成実行機能においては、他の計算機へのプロセス生成とプロセス移動を効率的に行えることが必要であることを述べ、プロセス変身機能を提案した。プロセス変身機能とは、実行プログラムの変更、開始位置の変更、および動作空間の変更を行える機能である。また、この機能を ***Tender*** に実現し、プロセス移動処理の基本性能を評価した。同一計算機内での動作空間の変更時間はプロセスの大きさの影響をほとんど受けないものの、他の計算機の動作空間への変更時間はプロセスの大きさに比例する。

残された課題として、異なる計算機にまたがるプロセス生成やプロセス移動における環境引継ぎの程度の検討がある。

参考文献

[1] Fred Douglis, M.Frans Kaashoek, John K. Ousterhout and Andrew S. Tanenbaum: "Comparison of Two Distributed Systems:Amoeba and Sprite", Computing Systems, Vol.4, No.4, pp.353–384 (1991)

[2] Michael Litzkow, Todd Tannenbaum, Jim Basney and Miron Livny: "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", Technical Report #1346, Computer Sciences Department, University of Wisconsin (1997)

[3] 白木原 敏雄 , 金井 達徳: "通信を行うプロセスの移送機能の設計と実装", 情報処理学会論文誌 , Vol.34 , No.6 , pp.1457–1467(1993)

[4] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler and Songnian Zhou: "Process Migration" , ACM Computing Surveys, Vol.32, No.3, pp.241–299 (2000)

[5] 谷口 秀夫 , 青木 義則 , 後藤 真孝 , 村上 大介 , 田端 利宏: "資源の独立化機構による ***Tender*** オペレーティングシステム" , 情報処理学会論文誌 , Vol.41 , No.12 , pp.3363–3374 (2000)

[6] 田端 利宏 , 谷口 秀夫: "プロセス再起動機能の提案と評価" , 情報処理学会コンピュータシステムシンポジウム , Vol.2000 , No.13 , pp.91–98 (2000)

[7] 谷口 秀夫 , 長嶋 直希 , 田端 利宏: "単一仮想記憶と多重仮想記憶を共存させたヘテロ仮想記憶の実現" , 情報処理学会研究会報告 , Vol.98 , No.33 , pp.87–94 (1998)

[8] 中島 耕太 , 下崎 誠 , 谷口 秀夫: "Myrinet を用いた高速データ通信機能の設計と実現" , 情報処理学会研究会報告 , Vol.2000 , No.43 , pp197–204 (2000)