位置透過に利用可能な構成要素を用いたプロセス変身機能

石井陽介[†] 谷口秀夫[††]

分散環境において，プロセス生成やプロセス移動を行うと，システム内の計算機間に依存関係が生じ，システム内の動的な機能変更を困難にすることが多い．そこで，本論文では，分散環境において，計算機間に依存関係を起こすことなく，ステートレスにプロセスを操作可能なプロセス構成法について述べ，それを用いて実現するプロセス変身機能について述べる．本構成法では，オペレーティングシステムが制御し管理する対象を資源として分離し，独立化させることで，プロセスの構成要素を細分割している．分割した資源を位置透過に利用可能にすることで，構成資源の存在場所に関係なくプロセスを構成することができる．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 3 つの機能からなる．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，プロセス変身処理の基本性能を示す．

A Process Transformation Mechanism Using Location-transparent Process Components

Yousuke Ishii[†] and Hideo Taniguchi[††]

In distributed environment process creation and process migration bring dependencies between the machines, which makes a dynamic change of the system difficult in many cases. In this paper we describe process composition method for distributed environment and process transformation mechanism. In this method the objects to be controlled and managed by operating system, which are called *resources*, are divided finely. The *resources* can be accessed location-transparently and statelessly without dependencies between the machines. Process can be composed of several *resources* regardless of their location. Our process transformation consists of three facilities, which changes components of a process, such as its program, start point and space. We describe the implementation of the process transformation mechanism on ***Tender*** operating system and show the performance.

1. はじめに

近年，複数の計算機を結んだ分散環境の利用が進んでいる．分散環境では，計算機の動的な追加や切り離しにより，システム内の構成を柔軟に変更できる．また，システム内に分散する計算機資源をプロセス間で共有することで，統合的な処理環境を実現可能である．統合的な処理環境を実現するためには，プロセスの生成，実行，消去といった一連の処理を，システム内で位置透過に実現し，プロセス間の通信や同期処理も実現する必要がある．特に，プロセスを位置透過に生成でき，かつ，実行途中のプロセスを動的に他の計算機に移動できる機能[1)]を実現すれば，システム内で負荷分散を実現することができ，非常に有益である．このように，分散環境の特徴を生かした処理を行うためには，システム内のプロセスの管理手法が重要になる．

分散環境において，オペレーティングシステム（以降，OS と略す）は，システム内で走行するプロセスに対して，そのプロセスが生成されてから消滅するまでの間，一貫した走行環境を提供する必要がある．特に，プロセスが移動した場合は，移動前にプロセスが行っていた処理を，移動後も継続して行えるようにしなければならない．このためには，当該プロセスが行う I/O 処理，プロセス間通信処理，およびシステムコール処理などを継続して行えることが望ましい．しかし，移動前に行っていた処理を移動後も継続して行えるようにするために，移動先計算機と移動元計算機との間に依存関係を残してしまう場合がある．この計

† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†† 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

算機間の依存関係は，負荷分散を行うために，システム内で多くのプロセスが動的に移動を繰り返す場合は，さらに複雑なものとなってしまう．計算機間の依存関係が複雑化すると，システム内の機能や構成を動的に変更することが難しくなり，柔軟にシステム内の構成を変更できるという分散環境の特徴を損なってしまう．そこで，分散環境において，システムを構成する計算機間に依存関係を起こすことなく，ステートレスにプロセスの生成，移動，および削除といった操作を行える方法についての検討を行った．本論文では，プロセスの構成要素に着目し，分散環境を指向するプロセス操作方法として，プロセス変身機能について述べる．

既存の多くの OS では，プロセスを 1 つの資源として扱っている．したがって，プロセスは多くの構成要素よりなっているため，プロセスという資源の単位が大きくなってしまう．この結果，プロセスの生成，移動，および削除といった操作を行う際．その処理にともなう処理負荷が大きくなってしまう．そこで，上記の問題を解決するために，従来のプロセスの構成要素を分割して細分化したものを資源とし，それぞれ個別に独立して扱えるようにする．また，分割された各資源は，システム内で位置透過に扱えるようにする．分割にともない，プロセスが持つ情報も分割し，それぞれ独立に存在させるようにする．これにより，各資源は位置透過に利用可能になり，プロセスを構成するために必要な資源を，その資源の存在場所に関係なく利用することが可能になる．

このようなプロセスの構成により，プロセスの各構成要素を個別に変更することが容易に実現できるようになる．プロセスの構成要素を変更することにより，当該プロセスの実行環境を変更できる．これにより，プロセスのチェックポイント/リスタートや他計算機上へのプロセス移動などを実現できる．たとえば，プロセス移動は，プロセスの動作空間を他計算機上の空間に変更することにより実現できる．さらに，各資源は位置透過に利用可能であるため，移動プロセスが移動前に利用していた資源を，移動後も引き続き利用して処理を行うことができる．したがって，プロセス移動処理時に特別な処理を行うことなく，ステートレスなプロセス移動も実現できる．

このように，プロセスの構成要素を変更できる機能を提供することは，プロセスが実行するプログラムの処理内容に合わせて，プロセスの実行環境を動的に変更できるようになるため，有益である．また，プロセスの各構成要素は，資源として独立しているため，プロセスの構成要素を変更する際は，変更対象となる要素のみを扱えるので，変更処理を効率化できる．さらに，1 回の呼び出し処理で複数の構成要素を変更できるようにすることにより，プロセスの複数の構成要素を変更できるサービスも実現できる．これにより，たとえば，プロセスのチェックポイント/リスタートとプロセス移動を組み合わせた統合サービスも実現できるようになる．そこで，プロセスの構成要素を変更する機能を，プロセス変身機能として実現する．具体的には，プロセス変身機能の基本機能として，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の 3 つを実現する．利用者に対しては，プロセス変身の各機能を単独で利用する形式を提供するだけでなく，複数の機能を組み合わせて利用する統合形式も提供する．この形式を基に，プロセス変身機能を実現することにより，プロセスのチェックポイント/リスタートやプロセス移動のような従来よりあるサービスだけでなく，その他のプロセスの構成要素を変更するサービス，ならびに複数の機能を統合した高度なサービスも実現する．

以降，2 章では，プロセスの構成要素を資源として分割し，各資源を位置透過でステートレスに利用できるプロセスの構成法について述べる．3 章では，プロセス変身機能について述べる．4 章では，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[2] における実現方式を示し，プロセス変身処理の基本性能を示す．その後，5 章で関連研究との比較について述べ，6 章で本論文をまとめる．

2. 分散環境を指向したプロセス構成法

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，様々な要素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．たとえば，内部資源には，仮想記憶空間，プロセッサ，レジスタ群などがあり，外部資源には，ファイル，ソケットなどがある．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

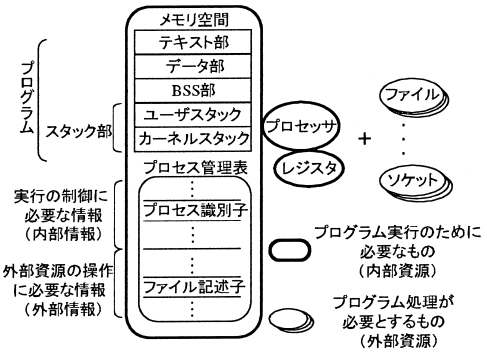

図 1 プロセスの構成要素
Fig. 1 Components of process.

行するときに利用する．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．

OS は，制御する対象を資源として管理している．ただし，既存の多くの OS では，OS が制御する対象をすべて資源として管理しているわけではない．プロセスを 1 つの資源として扱っている場合，プログラムやメモリ空間といったプロセスの構成要素は，資源として扱っていない．このため，プロセスを 1 つの資源として扱うと，プロセスという資源の単位が大きくなってしまう．プロセス資源の単位が大きくなると，たとえば，プロセス削除の際は，プログラムやメモリ空間のような，別の新たなプロセスが再利用できるプロセスの構成要素まで削除してしまう．さらに，プロセス生成の際は，再利用可能なプロセスの構成要素も，逐一新規に生成してしまう．つまり，プロセスを 1 つの資源として扱うと，プロセスの生成削除を行う際に，当該プロセスを構成するすべての要素の生成削除を行わなくてはならなくなる．このため，構成要素の一部分が再利用可能な場合があるにもかかわらず，逐一生成や削除を行う必要があり，プロセス生成削除の処理負荷が大きくなる．これにより，プロセスの生成削除を必要とするプロセス移動の処理負荷も大きくなってしまう．

一方，分散環境下では，位置透過なプロセス生成や，動的なプロセス移動により，システム内で負荷分散を行うことが考えられる．負荷分散を効率的に行うには，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．したがって，これらの処理を高速かつ省資源で行える必要がある．

上記の問題を解決するため，資源の分離と独立化を行う．つまり，既存の OS の資源を分割する．具体的には，プロセスの構成要素であるプログラムと仮想記憶空間と実メモリなどを資源化する．分割された資源は，それぞれ必要な情報を持たせて独立化させる．また，各資源には，資源識別子と資源名を付与し，かつ資源操作のインタフェースを統一する．さらに，資源識別子と資源名に，その資源の場所情報を含ませることにより，各資源を位置透過に扱えるようにする．このように，資源の分離と独立化を行うことにより，プロセスは，それ自身が利用する資源だけから構成することが可能になる．また，各資源は位置透過に利用可能になるので，利用資源の存在場所を意識することなくプロセスを構成することが可能になる．さらに，各計算機上で走行するプロセスは，利用している資源の管理について意識する必要はなくなる．そのうえ，各計算機上の OS は，自身の計算機上に存在する資源のみを管理すればよい．ただし，資源の位置透過な利用を可能にするために，当該計算機に対して，資源操作処理を依頼する機能は必要である．また，他計算機から資源操作処理を依頼された場合は，計算機間に依存関係を残さないように，ステートレスに処理を行うようにする．このように，他計算機からの資源操作処理をステートレスに行い，資源管理を各計算機ごとに局所化することで，システム内の計算機をそれぞれ独立に管理することができるうえに，システム内の動的な機能変更に対して柔軟に対応できるようになる．

このようなプロセスの構成により，プロセスの位置透過な生成処理を容易に実現できる．プロセス生成時には，プロセスとして実行するプログラムについての情報と，プロセスが利用する計算機，すなわち利用するプロセッサについての情報が必要になる．これらは資源として存在しているので，プロセス生成時に利用資源を直接指定することが可能である．また，利用資源を位置透過に指定し，利用することも可能であるため，ある計算機上に存在するプログラムを，別の計算機上に存在するプロセッサを利用して実行させることも容易に行える．この例を**図 2** に示す．図において，計算機 B 上のプロセスは，計算機 A 上のプログラムを実行し，計算機 A，ならびに計算機 C 上のファイルを操作し，さらに，計算機 C 上のプロセスとプロセス間通信を行っていることを表している．

プロセス移動処理についても，プロセスを構成する要素のうち，必要な要素のみを移動させるだけで実現できる．たとえば，プログラム資源は移動の必要がない．なぜなら，プログラム資源はプロセス生成時に，当該プロセスが利用するメモリ空間上に複写されるため，プロセス移動時には，プロセスが利用するメモリ

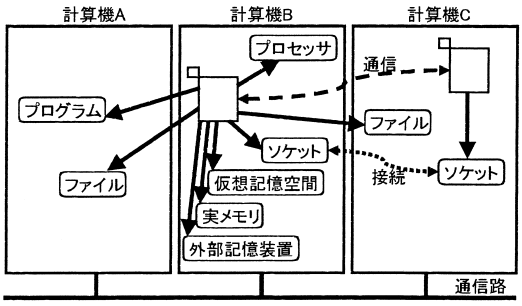

図 2 遠隔資源を利用したプロセスの構成
Fig. 2 Process composition using remote resources.

空間上のプログラムデータさえ移動させれば十分であるからである．また，プロセスの移動先において走行環境の再構築を行う必要はない．なぜなら，移動前のプロセスが，なんらかの資源を利用して処理を行っていた場合，そのプロセスが移動した後でも，利用している資源を位置透過に扱うことが可能なためである．したがって，プロセスが移動前に行っていた処理を，移動後も継続して行うことが可能である．

以上のように，本手法により，分散環境を指向したプロセス生成実行環境を実現することが可能である．ただし，本手法では，資源の分割を行っているため，プロセスを 1 つの資源として扱う方法と比べて，処理効率の低下が懸念される．処理効率を低下させる要因は，次の 2 つである．第 1 に，資源の分割により，資源を管理するモジュール間の呼び出し回数が増えるため，呼び出し処理に要するオーバヘッドが増大する．さらに，このモジュール呼び出し処理が計算機間にまたがって行われる場合は，その際の通信処理オーバヘッドも付加される．第 2 に，ローカル計算機内で資源操作処理を行う際，資源識別子，もしくは資源名に持たせている資源の場所情報より，操作対象計算機の判別処理を逐一行うために，判別処理に要するオーバヘッドが増大する．上記の 2 つについては，次の対処により，その負担を低減できる．前者については，資源を管理するモジュールを，モノリシックカーネルモデルを基にカーネル内で実現するようにする．これにより，モジュール間の呼び出し処理に要するオーバヘッドを抑制できる．また，後者については，操作の対象となる資源が，自計算機内に存在するのか否かを，各資源ごとに陽に指定できるようにする．これにより，陽にローカル指定した場合は，操作対象計算機の判別処理を省略できるようになり，処理効率の改善を図れる．

一方，資源の分割により，プロセスの生成削除処理から，当該プロセスの構成資源の生成削除処理を独立化でき，資源の生成や削除にともなう処理を高速化できるという利得が得られる．具体的には，構成資源を生成する際，再利用可能な資源が事前用意されている場合は，当該の事前用意されている資源を再利用する．また，構成資源を削除する際，当該の資源を再利用できるように保留しておくことにより，実際に当該の資源を解放しなくてもよくなる．これらの事前用意や保留を行う場合には，事前用意されている資源を検索する処理，ならびに資源を保留しておくための登録処理が必要になる．これらの検索処理や登録処理は，実際に資源を新規に生成する処理や解放する処理と比べて，処理負荷を小さくすることができる[3]．したがって，資源の分割によって生じる負担よりも，得られる利得を大きくすることができる．以上より，位置透過なプロセス生成や，動的なプロセス移動によって実現する負荷分散処理を，効率的に行うことが可能になる．

3. プロセス変身機能

3.1 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うち，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 3 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3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 3 つの機能の詳細について述べる．

3.2 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 1 つのプロセスとして逐次的に実行できる．

3.3 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 2 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[4] 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こる障害からの高速な回復処理を実現できる．

3.4 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 2 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，プロセス間で協調処理を行う際，頻繁にプロセス間通信を行って処理を行う場合，すなわち処理が緊密な場合は，一方のプロセスの動作空間を，他方のプロセスの空間へ変更して処理を行うことで，空間切替えによるオーバヘッドの削減が可能になる．また，処理の緊密さが変化する場合は，その処理の疎密度に合わせて，プロセスの動作空間を変更することも可能である．さらに，大きなデータを扱う処理を行う際，処理を行うプロセスが，データの存在する仮想記憶空間へ動作空間を変更して処理を行うというような，データ中心の処理を行うことが可能である．これを基にして，データの空間に存在できるプロセス数を 1 つに制限することにより，データへの排他処理も実現できる．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ともなうプロセス停止の回避が可能になる．

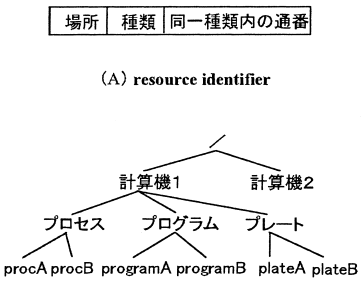

図 3 資源識別子と資源名

Fig. 3 Resource identifier and resource name.

3.5 複数の機能を組み合わせた利用

先に述べた 3 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. *Tender* における実現と評価

4.1 *Tender* オペレーティングシステム

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．***Tender*** において，資源操作を行う際は，資源インタフェース制御へ処理を依頼し，資源インタフェース制御が，各資源の操作を行う資源管理処理部を呼び出して処理を行っている．

資源には，**図 3** に示す資源識別子と資源名を付与する．資源識別子は，資源の場所と種類と同一種類内の通番を情報として有する数字である．資源名は，場所名と種類名と固有名からなる文字列である．場所とは，資源が存在する計算機を指す．場所を示す数字（以降，計算機番号と呼ぶ），ならびに名前（以降，計算機名と呼ぶ）は，システム内の計算機との対応関係とともに，計算機の場所情報表としてシステム内の特定計算機上で管理している．この計算機番号と計算機名の中には，陽にローカルを示す番号（0 番），ならびに名前（“tender”）を確保しており，これらを用いることで，資源を陽にローカル指定できるようにしている．

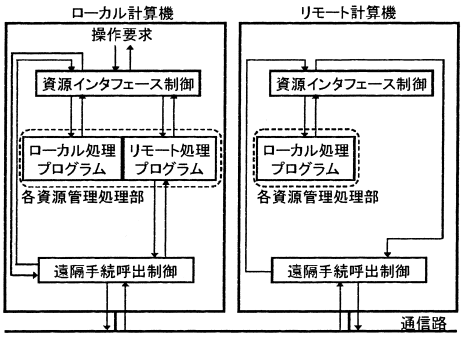

図 4 *Tender* における資源操作処理の様子
Fig. 4 Appearance of location-transparent resource operation on *Tender*.

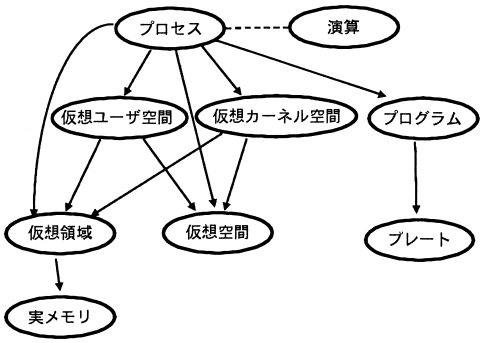

図 5 プロセスを構成する資源
Fig. 5 Component resources of process.

種類については，OS で数字や名前を規定している．同一種類内通番は OS が資源生成時に決定する．固有名は応用プログラムが指定する．資源識別子と資源名の変換機能は，資源インタフェース制御によって提供される．

また，***Tender*** では，ヘテロ仮想記憶[5] 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，1 つの仮想記憶空間内に 0 個以上のプロセスが存在できる．

4.2 資源操作方式

Tender では，位置透過な資源操作を実現している．ここでは，遠隔資源の操作を行うために，遠隔手続呼出制御を利用している．遠隔手続呼出制御は，遠隔計算機上にある手続きを呼び出す際，データの送受信を行う入出力とのインタフェース整合処理，自計算機内での手続呼出代行処理，および呼び出し結果の返送処理を制御している．

Tender における資源操作処理の様子を**図 4** に示す．資源操作を行うときは，ローカル計算機の資源インタフェース制御に処理を依頼する．資源インタフェース制御では，引数の資源名，もしくは資源識別子を調べて，当該の要求が陽にローカルを指定しているのか否かの判別を行う．陽にローカル計算機を指定している場合は，要求された処理を行う資源管理処理部のローカル処理プログラムを呼び出し，その戻り値を返す．逆に，陽にローカルを指定していない場合は，要求された処理を行う資源管理処理部のリモート処理プログラムを呼び出す．リモート処理プログラムは，リモート計算機に対して処理の依頼をするため，遠隔手続呼出制御を呼び出す．この遠隔手続呼出制御では，引数の資源名，もしくは資源識別子と，計算機の場所情報表との整合をとることにより，処理の依頼先計算機を判別する．この際，依頼先がローカル計算機である場合は，その処理要求を陽にローカル指定し直して，再度，ローカル計算機の資源インタフェース制御に処理の依頼を行う．逆に，依頼先がリモート計算機である場合は，通信路を介して，当該のリモート計算機に対して処理の依頼をする．リモート計算機では，依頼された処理要求を，ローカルを陽に指定するように変更して，自計算機の資源インタフェース制御に依頼する．資源インタフェース制御では，当該の要求は陽にローカルを指定していると判別し，要求された処理を行う資源管理処理部のローカル処理プログラムを呼び出し，その戻り値を返す．この戻り値は，処理依頼とは逆の手順で，依頼元のローカル計算機に返される．

4.3 プロセスの構成

Tender では，プロセスは多くの資源によって構成されている．プロセスを構成する資源を**図 5** に示す．矢印は，資源の依存関係を表している．ここで，資源「プロセス」とは，プロセス識別子とプロセス管理表からなり，OS がプログラムの動作を制御する単位になる．したがって，プロセス生成時には，資源「プロセス」が，生成先計算機上に生成され動作することになる．次に，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」とは，プロセスへのプロセッサ割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，プロセッサの割当てを受けて走行できる．

ここで，プロセスとメモリ関連資源との関係を**図 6** に示す．資源「仮想領域」とは，実メモリあるいは外

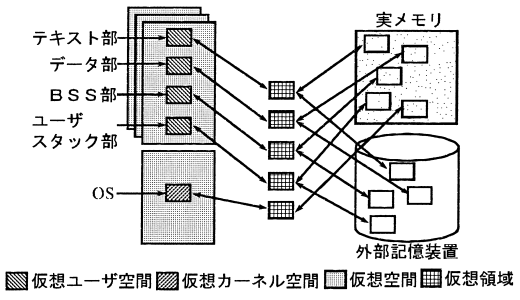

図 6 プロセスとメモリ管理関連の資源
Fig. 6 Resources of process and memory management.

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．資源「仮想空間」とは，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．資源「仮想カーネル空間」，ならびに資源「仮想ユーザ空間」とは，プロセッサが仮想アドレスによって，前者はカーネルモードのみ，後者はユーザモードでもアクセス可能な空間である．両者はともに，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．

上記のように，***Tender***では，プロセスの構成要素が資源として独立化している．また，各資源は，位置透過に利用可能である．したがって，遠隔計算機上に存在する資源を，その存在場所を意識することなしに，プロセスの構成要素として利用可能である．これにより，たとえば，遠隔計算機上に存在するプログラムを実行するプロセスを生成できる．

4.4 プロセス変身における特徴

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することで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，資源を分離し独立化させることにより，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ともな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 2 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 の機能を利用するときには，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などである．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ときに利用するものと，カーネルスタックのようにカーネルモードで走行するとき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とき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とき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ときに，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．

4.5 実 現 方 式

4.5.1 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 7** に示す．変更後のプログラムがすでに仮想領域にロードされている場

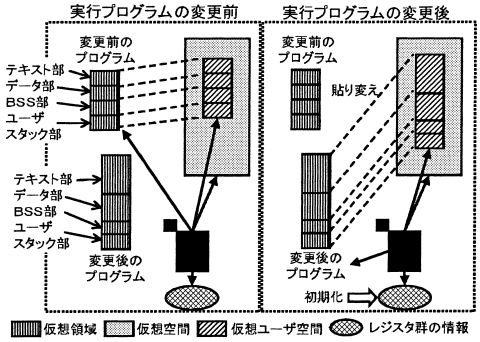

図 7 実行プログラムの変更
Fig. 7 Appearance of changing a program.

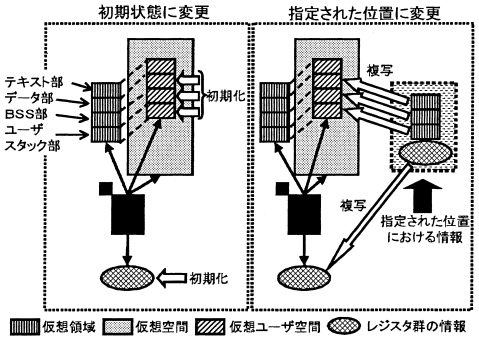

図 8 開始位置の変更
Fig. 8 Appearance of changing a start point.

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．

4.5.2 開始位置の変更

開始処理の変更の様子を**図 8** 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ときに，指定された位置から走行を開始できる．

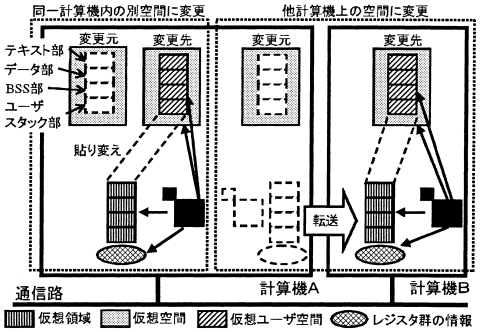

図 9 動作空間の変更
Fig. 9 Appearance of changing a space.

4.5.3 動作空間の変更

動作空間の変更の様子を**図 9** 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，まず初めに，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．

ここで，他計算機上の空間に変更を行うことで，他計算機上にプロセス移動を行う際，移動プロセスが外部資源を利用して処理を行っていた場合には，当該の外部資源をプロセス移動後も継続して利用できるようにしなくてはならない．しかし，ここでは，プロセス移動時に，当該の外部資源に対して，特に処理を行う必要はない．なぜなら，外部資源も資源インタフェース制御を介して位置透過な資源操作が可能なため，プロセス移動後も，資源識別子，もしくは資源名を用いて，当該資源に対する操作を行えるからである．したがって，プロセス移動処理時に，プロセス移動後も，移動プロセスは，当該の外部資源を利用した処理を継続して行うために，特別な処理を行う必要はなく，移動処理を簡略化できる．さらに，プロセス移動時に，

表 1 提供する基本インタフェース
Table 1 Basic interface.

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
Table 2 Integrated interface.

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.

移動プロセスが利用している外部資源の移動処理は必要ないため，プロセスの移動処理を簡略化でき，移動処理にかかる負荷を低減することもできる.

4.5.4 複数の機能を組み合わせた処理

プロセス変身機能は，3 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 2 つの場合は，処理の組合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

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，2 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.

4.5.5 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 3 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，1 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.

4.6 評価と考察

4.6.1 評価項目と測定環境

Tender に実現したプロセス変身処理の基本性能を評価するために，実測による評価を行った．測定を行った対象は，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における個別の処理時間である．また，複数の機能を組み合わせた場合における処

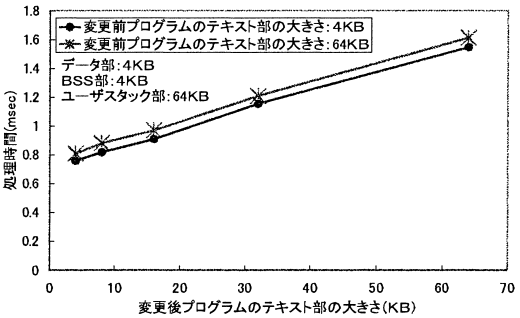

図 10 実行プログラムの変更処理時間とテキスト部の大きさとの関係
Fig. 10 Process transformation time (parameter is text size).

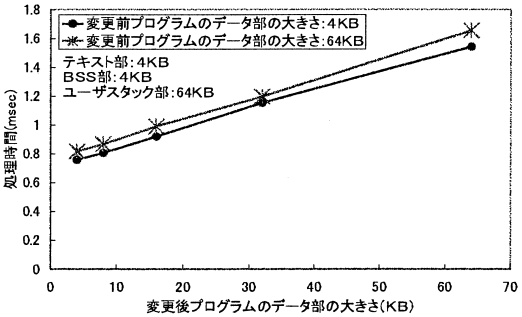

図 11 実行プログラムの変更処理時間とデータ部の大きさとの関係
Fig. 11 Process transformation time (parameter is data size).

理時間の測定も行った．測定では，表 2 の統合インタフェースを利用して，各処理を呼び出してから戻ってくるまでの時間を，ハードウェアカウンタにより計測した．測定は，Myrinet[6] により接続された PentiumII 450 MHz の計算機 2 台を用いて行った．Myrinet とは，1.28 Gbps の通信性能を持つ通信路のことである．ただし，測定では，資源の事前用意や保留を利用した資源の再利用は行っていない．

4.6.2 実行プログラムの変更処理

実行プログラムの変更処理時間の測定結果を**図 10**，ならびに**図 11** に示す．測定では，変更前後のプログラムの大きさと実行プログラムの変更処理時間との関係を調べた．

実行プログラムの変更処理時間と変更プログラムのテキスト部の大きさとの関係を図 10 に示す．図 10 より，実行プログラムの変更処理時間は，変更プログラムのテキスト部の大きさに比例して大きくなることが分かる．また，変更前プログラムのテキスト部の大きさよりも，変更後プログラムのテキスト部の大きさの方が，処理時間に大きな影響を与えていることも分かる．この原因としては，変更前プログラムが利用していた仮想ユーザ空間，ならびに仮想領域の削除処理よりも，変更後プログラムが利用する仮想ユーザ空間，ならびに仮想領域の生成処理の方が処理負荷が大きいうえ，変更後プログラムを読み込む処理も必要になるためと考えられる．

また，実行プログラムの変更処理時間と変更プログラムのデータ部の大きさとの関係を図 11 に示す．図 11 より，実行プログラムの変更処理時間は，変更プログラムのデータ部の大きさに比例して大きくなることが分かる．図 10 と図 11 を比較すると，変更プログラムの大きさが同じ場合は，同様な処理時間となっている．したがって，実行プログラムの変更処理時間は，テキスト部，またはデータ部という種類に関係なく，変更プログラムの大きさが影響していることが分かる．

4.6.3 開始位置の変更処理

開始位置の変更処理時間の測定結果を**図 12**，ならびに**図 13** に示す．測定では，開始位置を変更するプログラムの大きさと開始位置の変更処理時間との関係を調べた．測定を行った処理は，初期状態への変更処理，ならびに任意の指定された位置への変更処理の 2 つである．

開始位置の変更処理時間とテキスト部の大きさとの関係を図 12 に示す．図 12 より，開始位置の変更処理時間は，テキスト部の大きさに関係なくほぼ一定であることが分かる．この原因としては，開始位置の変更を行う際，対象プログラムのテキスト部に対して，特に処理を行う必要はないためと考えられる．

また，開始位置の変更処理時間とデータ部の大きさとの関係を図 13 に示す．図 13 より，開始位置の変更処理時間は，データ部の大きさに比例して大きくなることが分かる．この原因としては，初期状態に変更する場合，ならびに指定された位置に変更する場合のどちらにおいても，データ部の内容を読み込むためのデータ複写処理を行っているためと考えられる．

さらに，図 12，ならびに図 13 のどちらとも，初期状態に変更する場合の処理時間と比べて，指定された位置に変更する場合の処理時間の方が長くなることが分かる．この原因として，指定された位置に変更する処理を行う場合は，ユーザスタック部へのデータの複写処理を行うものの，初期状態に変更する場合は，ユーザスタック部の初期化処理のみを行うためと考えられる．

4.6.4 動作空間の変更処理

動作空間の変更処理時間の測定結果を**図 14**，ならびに**図 15** に示す．測定を行った処理は，同一計算機

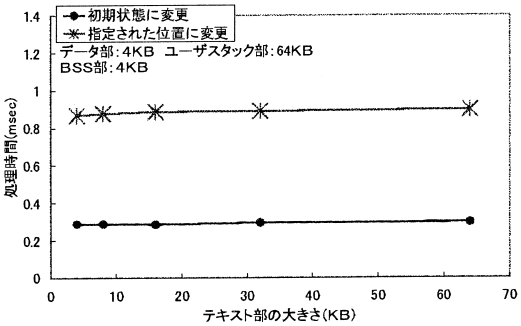

図 12 開始位置変更処理時間とテキスト部の大きさとの関係
Fig. 12 Process restart time (parameter is text size).

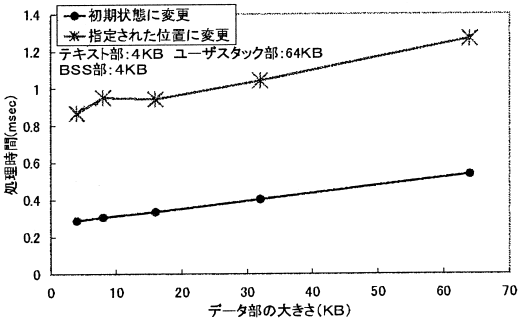

図 13 開始位置変更処理時間とデータ部の大きさとの関係
Fig. 13 Process restart time (parameter is data size).

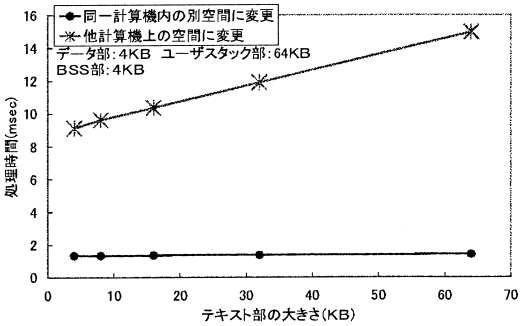

図 14 動作空間変更処理時間とテキスト部の大きさとの関係
Fig. 14 Process migration time (parameter is text size).

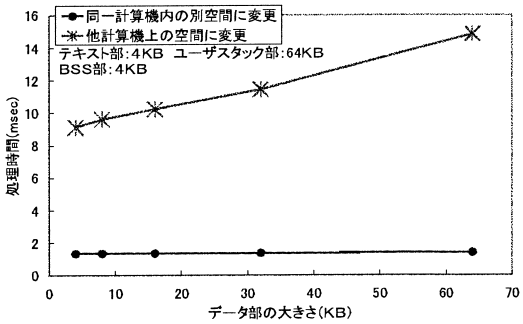

図 15 動作空間変更処理時間とデータ部の大きさとの関係
Fig. 15 Process migration time (parameter is data size).

内の別空間への動作空間変更処理，ならびに他計算機上の空間への動作空間変更処理の 2 つである．測定では，動作空間を変更するプログラムの大きさと動作空間の変更処理時間との関係を調べた．

動作空間の変更処理時間とテキスト部の大きさとの関係を図 14 に示す．図 14 より，同一計算機内の動作空間変更処理時間は，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分かる．一方，他計算機上の空間への動作空間変更処理時間は，同一計算機内の変更処理時間と比べて，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ことが分かる．この原因としては，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．

また，動作空間の変更処理時間とデータ部の大きさとの関係を図 15 に示す．図 14 と図 15 を比較すると，同一計算機内の変更処理時間，および他計算機上の空間への変更処理時間は，同様な値となっている．以上より，動作空間の変更処理時間は，変更先が同一計算機内，もしくは他計算機のどちらでも，動作空間を変更するプロセスが利用するプログラムの大きさに影響されることが分かる．

また，他計算機上への動作空間変更によるプロセス移動処理には，移動プロセスが利用しているすべてのプログラムデータを，移動先計算機に転送する方法を用いている．この方法は，移動先で利用しない可能性のあるデータも転送するため，データの転送効率は良くない．一方で，Copy-on-Reference を利用した転送法もある[1)]．この方法は，プロセス移動処理時にデータ転送を行わない．そのかわり，移動処理後に，移動先計算機で利用するプログラムデータを初めて参照するときに，移動先計算機へ転送する．たとえば，転送単位がページである場合は，参照データが存在するページを転送する．この方法は，プロセス移動処理時のデータの転送時間を短くし，転送効率を高める長所がある．しかし，短所として，プロセス移動処理後に付加的な転送処理を行う必要があるうえ，負荷の移動が緩慢になる．そのうえ，移動先プロセスは，移動元計算機への依存性を残してしまう．その点，今回利用している転送法は，プロセス移動処理時に，必要なデータ転送はすべて終了しているので，移動元計算機への依存関係を残さない．したがって，移動元計算機では，移動したプロセスを意識することなく処理を行えるうえ，移動元計算機の機能変更や電源断も容易に

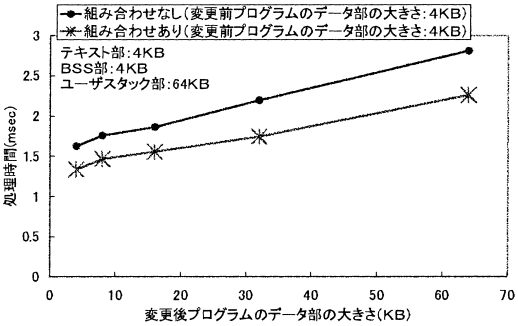

図 16 実行プログラム，および開始位置変更処理時間
Fig. 16 Process transformation and restart time.

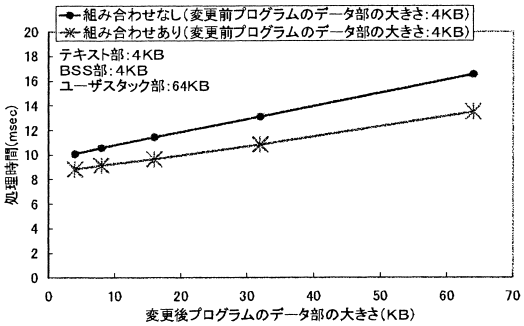

図 17 実行プログラム，開始位置，および動作空間変更処理時間
Fig. 17 Process transformation, restart and migration time.

行うことが可能になる．

4.6.5 複数の機能を組み合わせた処理

4.5.4 項において，プロセス変身処理は，複数の機能を組み合わせることによって処理を高速化できることを述べた．そこで，複数の機能を組み合わせることによる処理の高速化への効果について評価を行った．具体的には，4.5.4 項で述べた 2 つの場合について評価を行った．

はじめに，4.5.4 項の場合（1），すなわち実行プログラムの変更，ならびに指定された位置に変更する開始位置の変更を組み合わせて行った場合における処理時間を**図 16** に示す．測定では，変更後プログラムのデータ部の大きさと，変身処理時間との関係を調べた．図 16 には，組合せありとして，実行プログラムの変更，ならびに開始位置の変更を組み合わせて，変身処理を行った場合の処理時間を示す．また，組合せなしとして，2 つの変更処理を個別に呼び出して変身処理を行った場合の処理時間の合計を算出した値を示す．図 16 より，組合せありの方が，組合せなしと比べて処理を高速化できていることが分かる．また，組合せありの場合の処理時間 (y_1)，ならびに組合せなしの場合の処理時間 (y_2) を測定結果から近似した式を以下に示す．以下では，変更後プログラムのデータ部の大きさ（KB）を x としている．

$$y_1(\mu\text{sec}) = 15x + 1307 \quad (1)$$

$$y_2(\mu\text{sec}) = 19x + 1569 \quad (2)$$

式 (1)，ならびに式 (2) より，組合せありの場合のグラフの傾きは，組合せなしの場合よりも小さい．このため，組合せありの場合は，変更後プログラムのデータ部の大きさが大きくなればなるほど，処理の高速化への効果が大きいことも分かる．

次に，4.5.4 項の場合（2），すなわち他計算機上の空間へ動作空間を変更するとき，実行プログラムの変更，ならびに指定された位置に変更する開始位置の変更を組み合わせて行った場合における処理時間を**図 17** に示す．測定では，変更後プログラムのデータ部の大きさと，変身処理時間との関係を調べた．図 17 には，組合せありとして，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組み合わせて，変身処理を行った場合の処理時間を示す．また，組合せなしとして，3 つの変更処理を個別に呼び出して変身処理を行った場合の処理時間の合計を算出した値を示す．図 17 より，組合せありの方が，組合せなしと比べて処理を高速化できていることが分かる．また，組合せありの場合の処理時間 (y_3)，ならびに組合せなしの場合の処理時間 (y_4) を測定結果から近似した式を以下に示す．以下では，変更後プログラムのデータ部の大きさ（KB）を x としている．

$$y_3(\mu\text{sec}) = 77x + 8467 \quad (3)$$

$$y_4(\mu\text{sec}) = 107x + 9686 \quad (4)$$

式 (3)，ならびに式 (4) より，組合せありの場合のグラフの傾きは，組合せなしの場合よりも小さい．このため，組合せありの場合は，変更後プログラムのデータ部の大きさが大きくなればなるほど，処理の高速化への効果が大きいことも分かる．

5. 関 連 研 究

プロセス変身機能を実現する際に利用したプロセス構成法，ならびに位置透過な資源操作方式について，従来の方式と本論文で述べた方式との比較について述べる．V では，マイクロカーネルモデルを基にして，OS の制御対象を分割して管理している．具体的には，プロセス管理やメモリ管理などを，各計算機上に存在するカーネルサーバと呼ばれるサーバの形で実現している．資源は各サーバ内で管理され，各資源の文字列名は，資源を管理する各サーバが管理している．利用者は資源の文字列名をマルチキャストすることにより，

処理を依頼するサーバを特定する．サーバの特定時には，文字列名のプレフィックスとサーバの対応関係を保持するキャッシュを利用することで，通信回数を抑制している．ただし，サーバの特定処理時においては，キャッシュを利用しても，操作対象サーバを判別するための整合処理は必要になる．また，資源操作は，プロセス間通信を用いて当該サーバに依頼する形式を用いている．しかし，各サーバで管理する資源の粒度が大きいため，資源操作に要する負荷が大きい．逆に，資源の粒度が小さいと，各サーバ間の呼び出し処理，ならびにカーネルモードとユーザモード間の遷移に要するオーバヘッドが大きくなってしまう問題がある．

一方，本論文で述べた方式は，分割によって資源の粒度が小さいので，細かな資源操作を実現できるうえ，場所情報を持たせた資源識別子，ならびに資源名による位置透過な資源操作が可能である．また，資源操作要求を行う際，陽にローカル指定を行うことで，操作対象計算機を判別する整合処理を省略できるので，ローカル処理における処理効率の低下を防ぐことができる．さらに，OS の提供機能を，モノリシックカーネルモデルを基にカーネル内で実現することで，各サーバに相当する資源間の呼び出し処理に要するオーバヘッドを低減できる．そのうえ，資源の単独存在を可能にすることにより，資源の事前用意や保留による再利用で，資源の生成や削除にともなう処理を高速化できる．

次に，分散環境におけるステートレスなプロセス移動の実現方式について，従来の方式と本論文で述べた方式との比較について述べる．プロセス移動後において，移動前の処理を継続して処理できるようにする方式として，次の 3 つがある．

第 1 に，プロセス移動時に，移動元計算機におけるプロセスの走行環境を，移動先計算機に構築するという方式がある[7)~10)]．この場合，I/O 処理は，リダイレクト処理を行う必要がある．また，プロセス間通信処理は，通信路の保存や回復，移動処理時に送信されたメッセージの再送処理を行う必要がある．さらに，システムコール処理については，システムコール転送や遠隔手続呼出処理を行う必要がある．しかし，この方式は，移動プロセスが移動処理後においても，移動元計算機の走行環境を継続してひきずっている．そのうえ，移動元計算機では，内部の機能が他計算機から利用されるようになるため，その機能の一部を動的に変更することが難しくなる．したがって，柔軟にシステム内の構成を変更できるという分散環境の特徴を損なってしまう．

第 2 に，上記と同様なことを，OS のカーネルレベルではなく，ユーザレベルで実現する方式もある[11)~13)]．しかし，ユーザレベルでの対処は，処理速度が遅いという問題がある．

第 3 に，分散 OS により，システム内のどの計算機においても，同一の走行環境を提供するという方式もある[14)]．この場合，実行途中のプロセスがどの計算機に移動しても，移動後も継続して処理を行うことが可能である．しかし，この手法は，ローカル内の操作であっても，大域的な操作として扱われるため，処理効率が悪くなる問題がある．

一方，本論文で述べたプロセス変身によるプロセス移動方式では，プロセス移動時にステートレス性を確保するための処理を必要としない．これは，本論文で述べた構成要素を分割するプロセス構成法により，プロセスを構成する資源，ならびにプロセスがプログラム処理のために利用する資源は，各々が独立して存在可能であり，かつ位置透過に操作できるためである．したがって，プロセス移動後も，移動前に利用していた資源を移動先でも引き続き同じように操作できるため，移動前に移動プロセスが行っていた処理を継続して行える．

6. おわりに

分散環境を指向するプロセス操作法として，プロセス変身機能について述べた．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．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 3 つの機能からなる．利用者に対しては，プロセス変身の複数の機能を組み合わせて利用できる形式を提供している．このプロセス変身機能を実現するために，プロセスの構成要素を分割し，資源として独立化させる方式を利用した．この方式により，プロセスの各構成要素を個別に変更できるようになるうえ，変更対象となる資源のみを扱うことで，変更処理を効率化できる．また，システム内の各資源は，位置透過に利用できる操作方式を提供している．これにより，プロセス変身によって実現するプロセス移動では，ステートレスなプロセス移動を，プロセス移動処理時に特別な処理を行うことなく実現できる．

また，このプロセス変身機能を ***Tender*** に実現し，変身処理の基本性能を評価した．結果より，変身処理時間は，主に，変更されるプロセスが利用しているプログラムの大きさに比例することを示した．また，複数の機能を組み合わせた場合の処理時間についても評

価を行い，組合せによる効果として，処理を高速化できたことを明らかにした．

残された課題として，提案方式によって実現したプロセス変身機能における資源の再利用効果についての評価，プロセス変身機能を応用プログラムで利用したときの性能評価，および提案方式による位置透過なプロセス生成機能の実現と評価がある．

謝辞 本研究の一部は，日本学術振興会科学研究費補助金基盤研究（A）（2）（課題番号 15200002）による補助のもとで行われた．

参 考 文 献

1) Dejan, S.M., Fred, D., Yves, P., Richard, W. and Songnian, Z.: Process Migration, *ACM Computing Surveys*, Vol.32, No.3, pp.241–299 (2000).
2) 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).
3) 田端利宏，谷口秀夫：プロセス構成資源の効率的な再利用を目指した資源管理法，情報処理学会コンピュータシステムシンポジウム，Vol.2002, No.18, pp.21–28 (2002).
4) 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).
5) 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).
6) 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).
7) Cheriton, D.R.: The V Distributed System, *Comm. ACM*, Vol.31, No.3, pp.314–333 (1988).
8) 谷口秀夫：UNIX におけるプロセス制御機能のネットワーク化，情報処理学会マルチメディア通信と分散処理研究会，29-7 (1986).
9) Douglis, F. and Ousterhout, J.: Transparent Process Migration: Design Alternatives and the Sprite Implementation, *Software-Practice and Experience*, Vol.21, No.8, pp.757–785 (1991).
10) Damein, D.P., Andrzei, M.G., Michael, H. and Philip, J.: Performance Comparison of Process Migration with Remote Process Creation Mechanisms in RHODOS, *Proc. 16th International Conference on Distributed Computing Systems* (*ICDCS*), pp.554–561 (1996).
11) 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).
12) Michael, L., Todd, T., Jim, B. and Miron, L.: Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, Technical Report #1346, Computer Sciences Department, University of Wisconsin (1997).
13) Victor, C.Z., Barton, P.M. and Miron, L.: Process Hijacking, *Proc. 8th IEEE International Symposium on High Performance Distributed Computing* (*HPDC8*), pp.177–184 (1999).
14) 芝 公仁，大久保英嗣：分散オペレーティングシステム Solelc の設計と実装，電子情報通信学会論文誌，Vol.J84-D-I, No.6, pp.617–626 (2001).

(平成 14 年 8 月 19 日受付)
(平成 15 年 5 月 6 日採録)

石井 陽介（正会員）

平成 13 年九州大学工学部電気情報工学科卒業．平成 15 年同大学大学院システム情報科学府情報工学専攻修士課程修了．同年（株）日立製作所システム開発研究所入所．ストレージシステムの開発に従事．オペレーティングシステム，ストレージシステムに興味を持つ．

谷口 秀夫（正会員）

昭和 53 年九州大学工学部電子工学科卒業．昭和 55 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和 62 年同所主任研究員．昭和 63 年 NTT データ通信（株）開発本部移籍．平成 4 年同本部主幹技師．平成 5 年九州大学工学部助教授．平成 15 年岡山大学工学部教授．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書「オペレーティングシステム」（昭晃堂）等．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．