位置透過な資源操作方式によるプロセス生成機構

石井陽介[†] 谷口秀夫[††]

分散環境では，CPU やメモリといった計算機資源を複数利用して，負荷分散を実現できる．この負荷分散を効率的に実現するためには，プロセスを位置透過に再配置する機構が必要になる．そこで，本論文では，位置透過な処理を実現する資源操作方式を提案する．また，提案方式を基にしたプロセス生成機構について述べる．提案方式では，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，各資源を位置透過に同一形式で扱う．資源を分離する際，資源の粒度を小さくすることで，各資源操作の処理負荷を小さくする．本方式に基づき，プロセスの構成要素を細分割する．これにより，位置透過で効率良いプロセス生成処理を実現する．本論文では，これらの実現例として，***Tender***（The ENduring operating system for Distributed EnviRonment）オペレーティングシステムにおける実現方式を示し，資源操作処理，ならびにプロセス生成削除処理の性能を示す．

Process Creation Facility Based on Location-transparent Resource Operation Method

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

In distributed system a load distribution can be realized by using several CPUs and memories. To realize the load distribution a remote process creation and a process migration are needed. This paper proposes a resource operation method and describes a process creation facility based on the method. 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 uniformly and location-transparently. The processing time of resource operations becomes short. Based on this method, a process can be composed by the *resources*, and can be created efficiently. The proposed method and the facility are implemented on ***Tender*** operating system. This paper describes the implementation and shows 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し，統合的な処理環境を実現可能である．特に，分散環境では，複数の計算機資源を利用可能なため，負荷分散を行える．分散させる負荷には，CPU 負荷，ならびにメモリ負荷がある．CPU 負荷の分散により，CPU 利用率の低い計算機，つまり CPU 性能を持て余している計算機を有効利用することで，CPU 負荷が高い計算機上のプロセスの応答時間を短縮することが可能である．また，メモリ負荷の分散により，メモリ利用率の低い計算機を有効利用することで，仮想記憶機構を利用する際に起こる実メモリ空間と外部記憶装置間の入出力操作を削減することが可能である．これらの負荷分散を行うためには，システム内で，各計算機上の計算機資源を位置透過に，かつ効率良く操作できる方式を提供することが必要不可欠である．さらに，プロセスを静的に効率良く配置するためには，プロセスを位置透過に生成する機構が必要であり，また，動的に再配置するためには，プロセス移動機構[1)]が必要である．したがって，分散環境の特徴を生かすためには，システム内の資源操作方式，ならびにプロセスの管理手法が重要になる．そこで，本論文では，分散環境において，効率的に負荷分散を行う機構を実現するために，位置透過な資源操作方式，ならびにこの資源操作方式を基にした位置透過なプロセス生成機構について述べる．

† 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

既存の多くのオペレーティングシステム（以降，OSと呼ぶ）では，OSが制御し管理する対象の粒度が大きい．たとえば，既存の多くのOSでは，プロセスを1つの制御対象として扱っている．しかし，プロセスは，多くの要素により構成されているため，プロセスという制御対象の単位が大きくなり，プロセスの生成，移動，および削除の処理負荷が大きくなってしまう問題がある．したがって，たとえば，プロセスの再配置処理を行う際，その処理自身の負荷が大きくなるため，負荷分散の効果を低下させてしまうという問題がある．

そこで，本論文では，位置透過に，かつ効率良く操作可能な資源操作方式を提案する．本論文で提案する資源操作方式では，OSの制御対象を細分割することで，その粒度を小さくしている．また，分割した各々を資源として独立化させ，システム内で位置透過に扱えるようにする．これにより，効率的に負荷分散を行えるように，システム内に存在するどのプロセスからも，システム内の資源を位置透過に，かつ効率良く操作できるようする．また，従来は，個別に独立した操作ができなかった制御対象についても，資源として扱えるようにすることによって，効率の良い位置透過なプロセス生成処理を実現できるようにする．また，提案方式を基にしたプロセス生成機構について述べる．これにより，低い負荷でプロセス生成削除処理を実現し，効率的に負荷分散を行えるようになる．

本論文で提案する位置透過な資源操作方式，ならびに位置透過なプロセス生成機構は，***Tender***オペレーティングシステムに実現した．文献2)の***Tender***では，スタンドアロン環境において，資源の分離と独立化を基にした資源操作方式，ならびにプロセス生成実行機構を実現している．また，資源の分離と独立化の特徴を生かした機能として，プロセス生成削除の高速化を実現している．しかし，位置透過な資源操作処理は実現しておらず，位置透過なプロセス生成処理も行えない．そこで，本論文では，***Tender***で実現している資源操作方式を，位置透過に，かつ効率良く操作できるように拡張した．また，これを基にして，***Tender***に位置透過なプロセス生成機構を実現した．さらに，実現した処理について，基本性能の評価を行った．

以降，2章では，本論文で提案する位置透過な資源操作方式について述べる．3章では，提案方式を基にした位置透過なプロセス生成機構について述べる．4章では，提案機構の実現例として，***Tender***における実現方式を示し，評価結果を示す．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. 位置透過な資源操作方式

2.1 課題と対処

OSは，制御する対象を資源として管理している．以降では，資源とは，単独で存在し，資源を制御し管理する処理モジュールの外から資源の識別と操作ができるものとする．つまり，応用プログラム（以降，APと呼ぶ）からも，資源の識別と操作が可能である．

分散環境では，効率的に負荷分散を行えるように，システム内で，位置透過なプロセス生成処理を効率良く実現することが必要である．このためには，システム内に存在するどのプロセスからも，システム内の資源を位置透過に，かつ効率良く操作できるように方式を提供しなくてはならない．しかし，従来の方式では，Sprite[3]のように，入出力デバイスのような一部の計算機資源だけしか透過的に扱えない問題がある．また，V[4]のように，マイクロカーネルモデルを基にして，すべての資源を扱える方式もある．しかし，資源の粒度が大きいため，さらなる分割が可能な資源の生成削除を行う際に，構成要素の一部は再利用可能な場合があるにもかかわらず，逐一生成や削除が必要になり効率が悪い．さらに，資源操作時に必要な呼び出し処理に要するオーバヘッドが大きいという問題もある．したがって，高速で，かつ効率の良い位置透過な資源操作処理を提供する必要がある．

しかし，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．このため，資源として管理されていない対象については，その識別と操作が制限されてしまう．また，資源として管理されていない対象は，当該の対象を利用する資源が存在しなければ，単独で存在することもできない．たとえば，仮想記憶空間は，これを利用するプロセスの存在なしには存在できない．さらに，複数の構成要素からなる対象を1つの資源として扱うと，資源の粒度が大きくなる．このような粒度の大きい資源は，資源の生成や削除を行う際，構成要素の生成や削除を毎回個別に行う必要がある．このため，資源の生成や削除には，多大な処理負荷をともなってしまうという問題が生じてしまう．

上記の問題を解決するために，資源の分離と独立化を行う．つまり，既存OSの資源を細分割する．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，位置透過に識別と操作を行えるようにする．この資源の分離と独立化により，資源の粒度が小さくなるので，各資源操作に要する負荷を小さくできる．また，従来，識別や操作が制限され，独立して存在する

ことができなかった制御対象に対して，細かな資源操作が実現できる．

しかし，上記の対処法では，資源の細分割を行うため，資源を細分割しない方法と比べて，資源操作における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としては，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ことが考えられる．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．そこで，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．

一方，資源の分割により，複数の構成要素からなる資源の生成削除処理から，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，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することができる．これにより，特に，複数の要素より構成されるプロセスの生成削除処理を高速化することができる．たとえば，仮想記憶空間が単独で存在できれば，プロセス生成処理より以前の負荷が小さいときに，仮想記憶空間を作成しておくことにより，プロセス生成処理を高速化できる．

2.2 対処の具体化

位置透過な資源操作を実現するためには，システム内の全資源を一意に識別でき，同一形式で操作できる必要がある．

まず，資源識別のために，全資源に対して，文字列による資源名，ならびに数字による資源識別子を付与する．これらを両方利用することにより，AP を記述する際の利便性を高め，かつ操作対象資源の特定処理を高速化することができる．このため，資源名と資源識別子の対応関係を管理し，一方を他方へと変換できるようにする．また，システム内の資源を一意に識別できるようにするために，各資源に付与する資源名と資源識別子の中に，当該資源の場所情報を持たせる．この資源の場所情報については，システム内の計算機との対応関係をシステム内の特定計算機上で管理し，システム内で共有して利用するようにする．対応関係を特定計算機で管理することにより，管理情報の追加や更新作業を一元化し，情報の一貫性を保持する．

次に，位置透過な資源操作のために，資源操作要求

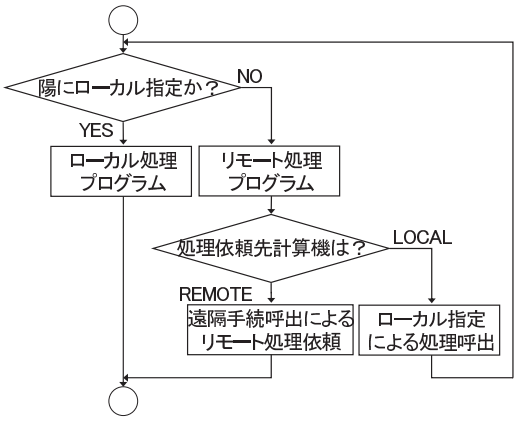

図 1 位置透過な資源操作処理の様子
Fig. 1 Appearance of location-transparent resource operation.

を行う計算機（以降，ローカル計算機と呼ぶ）内における資源操作のインタフェースと，別の計算機（以降，リモート計算機と呼ぶ）上の資源操作のインタフェースを同一形式にする．ここで，操作対象計算機の判別には，資源名と資源識別子の中に含まれる資源の場所情報を利用する．ただし，資源操作を要求する際，操作対象計算機を逐一指定することになるため，ローカル計算機内で行う資源操作の処理効率が悪くなるという問題がある．そこで，操作対象資源が，ローカル内に存在するのか否かを陽に指定できるようにして，かつ資源操作処理を**図 1** のようにする．図 1 における処理の流れでは，陽にローカル指定していない場合のみ，処理依頼先計算機の判別処理を行うようにしている．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が不要になるため，ローカル計算機内における資源操作の処理効率を改善できる．

3. 位置透過なプロセス生成機構

3.1 プロセスの構成

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，多くの要素から構成されている．プロセスの構成要素を**図 2** 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

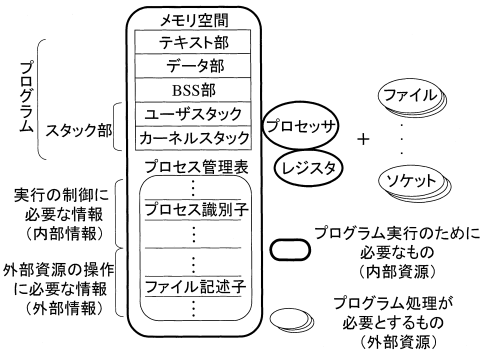

図 2 プロセスの構成要素
Fig. 2 Components of process.

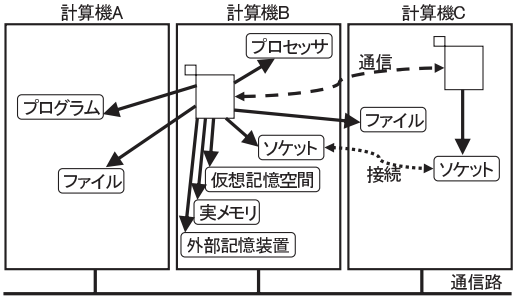

図 3 遠隔資源を利用したプロセスの構成
Fig. 3 Process composition using remote resources.

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．

3.2 既存 OS の問題点

既存の多くの OS においては，プロセスを 1 つの資源として扱っており，プログラムやメモリ空間といったプロセスの構成要素を資源として扱っていない．このため，プロセスという資源の単位が大きくなり，プロセスの生成，移動，および削除にともなう処理負荷が大きくなってしまう問題がある．

分散環境下では，位置透過なプロセス生成により，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，低い負荷でプロセス生成処理を行う必要がある．したがって，位置透過なプロセス生成処理を，高速かつ省資源に行う必要がある．また，プロセスの構成要素の中には，プログラムのようにシステム内の計算機間で共有可能な構成要素も存在する．したがって，共有可能な構成要素を，計算機間で効率良く共有できる手段を提供する必要もある．

3.3 対 処 法

3.2 節で示した問題を，前章で述べた資源の分離と独立化を行うことにより解決する．つまり，プロセスを細分割し，各々を資源として独立化させる．たとえば，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を資源化する．これにより，プロセスは，それ自身が利用する資源を自由に組み合わせることによって構成可能になる．また，各資源は位置透過に利用可能なため，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．これにより，計算機間で共有可能な資源を用いてプロセスを構成することもできるため，共有可能な資源を効率良く利用できるようになる．ただし，プロセス構成資源が複数の計算機に散在することにより実行時のオーバヘッド増大を招き，かえって負荷分散効率が下がることも考えられる．そこで，実行時のオーバヘッド増大を防ぐために，プロセスの構成資源として利用可能な資源の存在場所に，一定の制約を設ける必要がある．このため，当該プロセスが利用するプロセッサ資源，レジスタ資源，およびメモリ資源については，ともに同一計算機上に存在しなければならないという制約を設けることにする．

本方式により，プロセスを粒度が小さい資源で構成できる．このため，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させることができる．したがって，資源の再利用[2] により，構成資源の生成削除処理を簡略化することができるため，プロセス生成削除処理を高速化することができる．

位置透過なプロセス生成を行う場合は，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．本方式では，これらを資源として直接指定し，利用することが可能であり，プロセスの位置透過な生成を実現することができる．また，プロセスがプログラム実行時に利用する資源を位置透過に指定し，利用することも可能である．このため，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可能である．この例を**図 3** に示す．図 3 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．

さらに，本方式により，プロセス移動処理も高速化できる．具体的には，移動先で利用する資源のみを移動させることにより，プロセス移動を実現できる．これにより，移動処理にともなう資源生成削除処理，ならびに通信路を介したデータ転送量を削減することができる．

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*** では，資源を管理しているプログラム部分（以降，資源管理処理部と呼ぶ）を独立化させるため，表プログラム構造という機構を持つ．表プログラム構造とは，プログラム部品と，プログラム部品へのポインタを持つプログラムポインタ表からなる．プログラムポインタ表の行要素と列要素は，操作する資源の種類と操作内容に対応している．資源管理処理部は，資源への操作を，資源の生成（open 系），削除（close 系），入力（read 系），出力（write 系），および制御（control 系）の 5 つに分類し，各々をプログラム部品として実現している．プログラムポインタ表は，***Tender*** 特有の部分である資源インタフェース制御によって管理される．資源インタフェース制御は，プログラム部品の登録，削除，および変更を行う機能を提供し，プログラム部品への呼び出しを制御している．つまり，各資源の操作に必要な資源管理処理部の呼び出しは，資源インタフェース制御へ依頼し，資源インタフェース制御がプログラムポインタ表を用いてプログラム部品を呼び出すようにしている．

資源には，**図 4** 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．計算機番号と計算機名の中には，ローカルを示す番号，ならびに名前を確保している．具体的には，システム内のすべての計算機において，ローカルを示す番号として 0 番を，また，ローカルを示す名前として “tender” を確保している．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．種類については，OS で数字や名前を

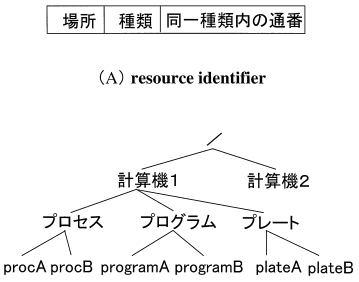

図 4 資源識別子と資源名
Fig. 4 Resource identifier and resource name.

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 AP が指定する．

ここで，資源識別子，ならびに資源名については，以下に述べる方式で管理を行う．まず，OS では，資源識別子と資源名の対応関係を管理し，一方を他方へと変換する機能を提供するようにする．具体的に，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．ただし，資源に付与する資源識別子，ならびに資源名の中には，その資源の存在場所を示す情報も格納している．このため，資源の存在場所が変わる場合，すなわち資源が移動する場合は，当該資源の資源識別子，ならびに資源名を変更する必要がある．ここで，資源の移動については，AP，もしくはミドルウェアに相当する OS 核外より要求を受けて行うことを想定している．このため，資源の移動を要求する際には，OS 核外で資源の移動先を指定することになる．したがって，移動を要求された資源の資源識別子，ならびに資源名の変更内容については，OS 核外で把握できるため，資源の移動によって生じる問題に対しては，OS 核外で必要な対処を行うようにする．また，システム内の計算機が動的に削除された場合については，当該計算機上に存在している資源を利用しているプロセスに対して，OS 側から特に通知は行わない．したがって，この場合についても，OS 核外で必要な対処を行うようにする．具体的には，計算機の削除を行う前に，当該計算機上にある資源の中で，削除後も必要になる資源については，別計算機上へ移動させる処理をあらかじめ行っておくことが考えられる．

4.2 資源操作方式

Tender では，資源の場所情報を含んだ資源名，もしくは資源識別子を利用し，資源インタフェース制御を介することによって，位置透過な資源操作を可能にしている．ここでは，遠隔資源の操作を行うために，

表 1 資源操作のための提供インタフェース
Table 1 Interfaces for resource operation.

操作の種類	形式	機能（pid：共通引数，処理要求プロセスのプロセス識別子）
open	open_rsc(rsc_name, pid, args, mod)	引数 args を利用して資源を生成し，操作権 mod と資源名 rsc_name を付与する．
非 open	***_rsc(rid, pid, args)	引数 args を利用して資源識別子 rid で指す資源に***操作を行う．ここでは，*** = “close, read, write, control”．

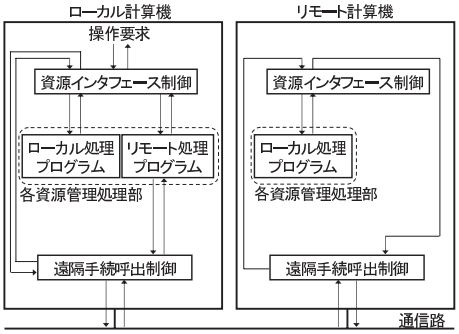

図 5 ***Tender*** における資源操作処理の様子
Fig. 5 Appearance of location-transparent resource operation on ***Tender***.

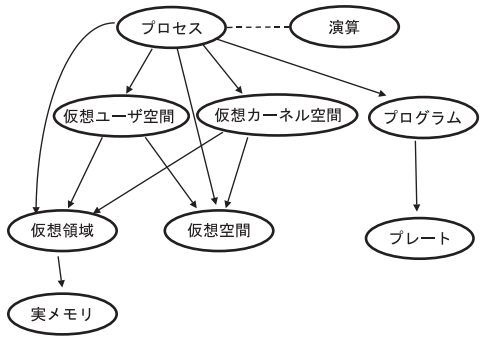

図 6 プロセスを構成する資源
Fig. 6 Component resources of process.

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 5** に示す．資源操作は，**表 1** の提供インタフェースを利用して，ローカル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，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取ることにより，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 プロセス生成機構

4.3.1 プロセスの構成

Tender では，資源の分離と独立化により，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 6** 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，演算を関連付けることで，

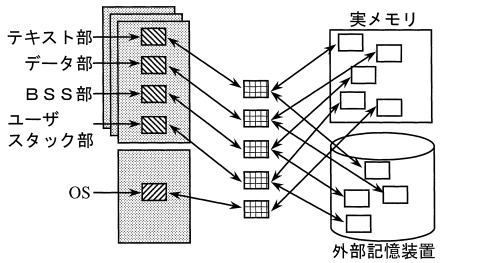

図 7 プロセスとメモリ管理関連の資源
Fig. 7 Resources of process and memory management.

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 7** に示す．資源「仮想領域」とは，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．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．

4.3.2 プロセス生成実行の特徴

Tender では，プロセスの生成実行に関して，次に述べる 3 つの特徴がある．

第 1 に，資源の位置透過な利用が可能なため，遠隔資源を利用したプロセスの構成が可能である．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利用してプロセスを生成できる．

第 2 に，資源の分離と独立化により，資源の事前用意や保留が可能になっている．これにより，資源の生成や削除に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である．たとえば，プロセスへのプロセッサ割当て単位を資源「演算」として資源化し，プロセスとは独立に存在させ，プロセスに演算を関連付けることでプロセスの実行が開始される．これにより，あらかじめ，プロセスや演算を作り置きしておくことにより，プロセスの生成実行処理を高速化することが可能である．

第 3 に，走行途中のユーザプロセスを遠隔計算機上に移動させることが可能である．***Tender*** におけるプロセス移動は，まず，移動先計算機上にプロセスを生成し，生成したプロセスをプロセス変身機能[5] を用いて，移動プロセスへと変身させることで実現している．このプロセス移動により，動的に負荷分散を行うことを可能にしている．

表 2 プロセスの生成形態
Table 2 Patterns of process creation.

通番	名称	プロセス生成先	プログラム存在先	備考
(1)	LL	ローカル	ローカル	
(2)	RR	リモート	リモート	各リモートは同一
(3)	LR	ローカル	リモート	
(4)	RL	リモート	ローカル	
(5)	RR′	リモート	リモート	各リモートは別々

4.3.3 プロセス生成法

位置透過なプロセス生成を実現するためには，生成したプロセスが走行する計算機，ならびにプロセスとして実行するプログラムを，位置透過に指定できるようにする必要がある．プロセスの生成形態としては，**表 2** のように 5 つの形態が考えられる．これらの形態のうち，形態（LL），形態（RR），および形態（LR）を実現すれば，残りの形態（RL），ならびに形態（RR′）についても，それらを組み合わせて利用することで実現可能である．具体的には，形態（RL）は，形態（LR）の処理においてプロセス生成処理要求を行う際，形態（RR）の処理で利用するリモートへのプロセス生成処理要求を行うことで実現できる．また，形態（RR′）は，形態（RR）の処理においてプログラムに関する処理を行う際，形態（LR）の処理で利用するリモートへのプログラム処理要求を行うことで実現できる．

また，4.2 節で述べた位置透過な資源操作方式を用いることにより，図 6 で示した各構成資源については，システム内のどの計算機上に存在している資源でも，プロセスの構成資源として利用することが可能である．ただし，プロセス構成資源が複数の計算機に散在することにより実行時のオーバヘッド増大を招くことも考えられる．そこで，実行時のオーバヘッド増大を防ぐために，プロセスの構成資源として利用可能な資源の存在場所に，一定の制約を設けている．具体的には，図 6 にある資源「演算」，資源「仮想空間」，資源「仮想カーネル空間」，資源「仮想ユーザ空間」，および資源「実メモリ」については，同一計算機上に存在しなければならない．ただし，メモリ関連資源の 1 つである資源「仮想領域」については，他のメモリ関連資源とは異なり，別計算機上に存在している資源を利用することも可能である．この資源「仮想領域」に

表 3 プロセス生成，実行，および削除のために資源管理処理部が提供するインタフェース
Table 3 Interfaces for process creation, execution and deletion.

通番	操作内容	形式	機能
(1)	プロセス生成	creat_proc(rsc_name, plateid, argv, vmid)	plateid で指すロードモジュールプレートをプログラムとして持つプロセスを生成し，資源名 rsc_name を付与する．argv は，プログラムへの引数を指定する．vmid は，プロセスが利用する仮想空間を指定する（通番 0 の場合は仮想空間を新規に生成）．
(2)	演算生成	creat_execution(rsc_name, mips)	演算の程度 mips を持つ演算を 1 つ生成し，資源名 rsc_name を付与する．
(3)	演算関連付け	attach_execution(execid, pid)	execid で指す演算を，pid で指すプロセスに関連付ける．
(4)	プロセス削除	delete_proc(pid, rflag)	rflag で指定された資源を保留し，プロセスを消滅させる．

表 4 プロセス生成，実行，および削除を行うカーネルコールインタフェース
Table 4 Kernel call interfaces for process creation, execution and deletion.

通番	操作内容	形式	機能
(1)	プロセス生成	proccreate(proc_name, plate_name, argv, vmid)	plate_name で指すプレートをプログラムとして実行するプロセスを生成し，資源名 proc_name を付与する．argv は，プログラムへの引数を指定する．vmid は，プロセスが利用する仮想空間を指定する（通番 0 の場合は仮想空間を新規に生成）．
(2)	プロセス実行	procgetexec(pid, mips)	演算の程度 mips を持つ演算を 1 つ生成し，pid で指すプロセスに生成した演算を関連付ける．
(3)	プロセス削除	procdelete(pid)	pid で指すプロセスとその構成資源を削除する．

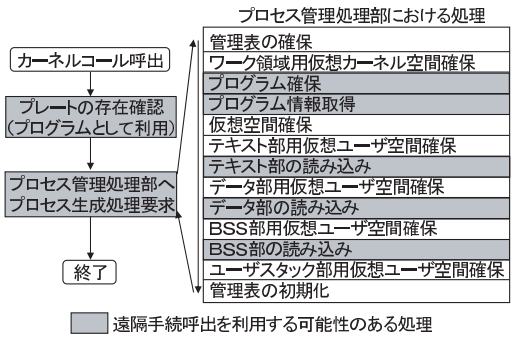

図 8 プロセス生成処理の流れ
Fig. 8 A flow of process creation.

ついては，複数の計算機間で共有して利用することにより，分散共有メモリ[6] として利用することができる．

ここで，プロセスを生成するために，OS 内部で行う処理が利用するインタフェースを**表 3** の通番（1）に示す．このインタフェースは，プロセス管理処理部が提供している．また，プロセスを生成するために，ユーザが利用するカーネルコールのインタフェースを**表 4** の通番（1）に示す．さらに，そのカーネルコールを呼び出すことによって実行されるプロセス生成処理の流れを**図 8** に示す．図 8 において，カーネルコールの引数 plate_name を利用して，プログラムとして利用するプレートの存在を確認し，そのプレート識別子を獲得する．ここで，当該プレートがリモートに存在する場合は，遠隔手続呼出制御を利用する．その後，プロセス管理処理部に対してプロセス生成処理を要求する．なお，生成先がリモートの場合も，遠隔手続呼出制御を利用する．プロセス管理処理部における処理では，プログラムがリモートに存在する場合，プログラムに関する処理を遠隔手続呼出制御を利用して行う．

4.3.4 プロセス実行法

Tender において，プロセスを実行するためには，演算を生成し，それを当該プロセスに関連付ける必要がある．ここで，演算の生成と，関連付けのために，OS 内部で行う処理が利用するインタフェースを表 3 の通番（2），および通番（3）に示す．これらのインタフェースは，演算管理処理部が提供している．また，演算を生成し，プロセスへ関連付けるために，ユーザが利用するカーネルコールのインタフェースを表 4 の通番（2）に示す．表 3，ならびに表 4 に示すインタフェースを利用することにより，演算の生成，ならびに演算のプロセスへの関連付けを位置透過に実行できる．

4.3.5 プロセス削除法

プロセスを削除するために，OS 内部で行う処理が利用するインタフェースを表 3 の通番（4）に示す．このインタフェースは，プロセス管理処理部が提供している．ここでは，引数 rflag で指定したプロセスの構成資源を，再利用可能資源管理表に登録する[7]．こ

の管理表に登録された資源は，以降で，プロセスを生成する際に構成資源として再利用されることになる．また，プロセスを削除するために，ユーザが利用するカーネルコールインタフェースを表 4 の通番（3）に示す．表 3，ならびに表 4 に示すインタフェースを利用することにより，プロセスの削除を位置透過に実行できる．

4.4 評価と考察

4.4.1 評価項目と測定環境

Tender に実現した処理について，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．資源操作処理，ならびにプロセス生成削除処理に要する処理時間を測定の対象とすることにより，これらを利用する負荷分散処理に要する処理時間を見積もることができる．このため，測定を行った処理対象は，資源操作処理時間，プロセス生成処理時間，およびプロセス削除処理時間とした．測定環境として，計算機（PentiumII 450 MHz）を 2 台，通信路に Myrinet [8]（1.28 Gbps）を利用した．測定では，ディスク I/O 処理による測定結果への影響を除外するために，ディスク I/O は発生しない環境とした．また，利用した OS は，***Tender*** ver. 7.1 である．ただし，***Tender*** ver. 7.1 では，システム内の特定計算機上で管理している計算機の場所情報表を共有して利用する機構を実現していない．このため，資源操作処理の評価において，計算機の場所情報表の獲得処理を除外した処理時間の評価を行えるようにするために，各計算機上に計算機の場所情報表を持たせた状態で行った．また，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．資源の再利用効果については，4.4.4 項で述べる．

4.4.2 資源操作処理

資源操作処理時間の測定結果を**表 5** に示す．測定を行った資源操作の処理内容は，プロセス生成処理（open 系），ならびにプロセス削除処理（close 系）である．また，処理形態は，陽にローカル指定したローカル処理（L1），陽にローカル指定しないローカル処理（L2），およびリモート処理（R）である．

測定結果より，形態（L1）と形態（L2）の処理時間には大差がない．これは，システム内の特定計算機で管理している計算機の場所情報表を，各計算機上にも持たせた状態で測定を行ったためである．したがって，実際には，形態（L2），ならびに形態（R）の処理時間には，計算機の場所情報表の獲得処理時間も付加される．この点を考慮にいれると，形態（L1）の処理時間は，形態（L2）の処理時間と比べて高速になることが分かる．また，形態（L1）と形態（L2）の処理時間の差に着目すると，プロセス生成処理時における差よりも，プロセス削除処理時における差の方が小さい．この理由は，プロセス生成時には，文字列による資源名を用いて資源識別を行っているのに対して，プロセス削除時には，数字による資源識別子を用いて資源識別を行っているためである．上記と同様なことは，形態（L2）と形態（R）の処理時間の差からも分かる．

表 5 資源操作処理時間（μsec.）
Table 5 Resource operation time (μsec.).

処理内容/処理形態	（L1）	（L2）	（R）
プロセス生成	1,801	1,810	2,533
プロセス削除	281	285	966

<テキスト部：4KB，データ部：なし，BSS 部：なし>

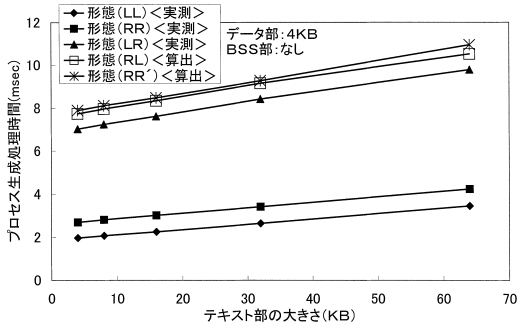

図 9 プロセス生成処理時間
Fig. 9 Process creation time.

4.4.3 プロセス生成削除処理

図 9 に，表 4 の通番（1）に示しているカーネルコールを利用したプロセス生成処理時間を示す．実測による測定対象は，表 2 の形態（LL），形態（RR），および形態（LR）である．測定では，生成プロセスが実行するためのプログラムを，テキスト部の大きさを可変にして，プログラムの大きさとプロセス生成処理時間との関係を調べた．なお，形態（RL），ならびに形態（RR′）については，以降で述べる．

測定結果より，プロセス生成処理時間は，プログラムの大きさが同じ場合，形態（LL）が最も高速で，形態（RR），形態（LR）の順に遅くなる．この原因として，各処理における遠隔手続呼出制御の利用回数が考えられる．ちなみに，形態（LL）の利用回数は 0 回，形態（RR）の利用回数は 1 回（プロセス生成），形態（LR）の利用回数は 6 回（プレートの存在確認，プログラム確保，プログラム情報獲得，テキスト部の読み込み，データ部の読み込み，および BSS 部の読み込み）である．ただし，***Tender*** ver. 7.1 では，プログラムのデータ部，または BSS 部が存在しない場合でも，各部の読み込み処理を呼び出している．このため，

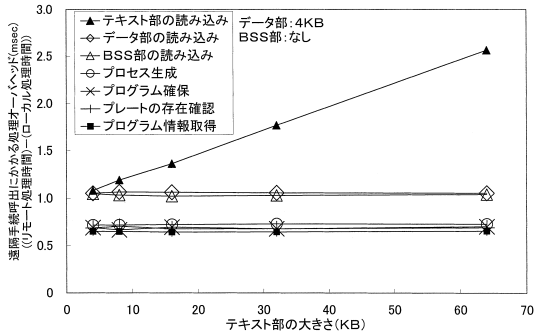

図 10 プロセス生成時に利用する遠隔手続呼出の処理オーバヘッド
Fig. 10 Overhead for remote procedure call in process creation.

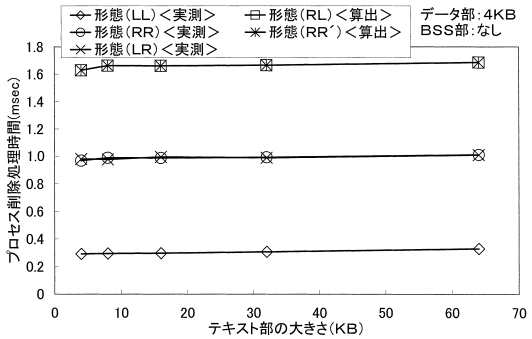

図 11 プロセス削除処理時間
Fig. 11 Process deletion time.

形態（LR）におけるプロセス生成時には，データ部やBSS 部が存在しなくても，それらの読み込み処理を呼び出すために遠隔手続呼出制御を呼び出している．

次に，グラフの傾きは，形態（LL）と形態（RR）はほぼ同じなのに対して，形態（LR）は他と比べて大きい．この原因として，形態（LR）では，通信路を介したプログラムデータの転送を行っているので，プログラムの大きさの増加に比例して，その転送時間も増加しているためと考えられる．

また，表 2 の形態（RL），ならびに形態（RR′）におけるプロセス生成処理については，***Tender*** ver. 7.1 では処理を実現していない．そこで，遠隔手続呼出処理を個別に実測することにより，これらの処理時間を推定する．具体的には，形態（RL），ならびに形態（RR′）のプロセス生成処理時間として想定できる値を算出するために，プロセス生成処理の中で，図 8 で示した遠隔手続呼出を利用する可能性のある処理について，それぞれの処理で遠隔手続呼出の処理オーバヘッドを実測により調べた．

図 10 に，プロセス生成時に利用する遠隔手続呼出の処理オーバヘッドの測定結果を示す．図 10 には，図 8 で示した遠隔手続呼出を利用する可能性のある処理について，遠隔手続呼出を利用してリモートで処理を行った場合の処理時間から，ローカル内で処理を行った場合の処理時間を引いたものを，遠隔手続呼出の処理オーバヘッドとして示している．図 10 より，プレート存在確認，プロセス生成，プログラム獲得，およびプログラム情報取得を行う場合，遠隔手続呼出の処理オーバヘッドは，プログラムの大きさに関係なく，ほぼ一定であることが分かる．また，テキスト部の読み込みを行う場合，遠隔手続呼出の処理オーバヘッドは，読み込むプログラムの大きさに比例して大きくなることが分かる．このことは，データ部の読み込み，ならびに BSS 部の読み込みを行う場合にも当てはまる．ここで，テキスト部，データ部，および BSS 部の読み込みを行う場合，プロセス生成などの場合と比べて遠隔手続呼出の処理オーバヘッドが大きいのは，通信処理時において，読み込むプログラムデータを格納するバッファの確保処理が別途必要なためと考えられる．

図 9 の実測結果，ならびに図 10 の結果より，表 2 の形態（RL），ならびに形態（RR′）におけるプロセス生成処理時間として想定できる値を算出した．ここで，形態（RL）におけるプロセス生成処理時間については，形態（LR）におけるプロセス生成処理要求を，リモートに対して行うことで実現できる．このため，形態（RL）の生成処理時間は，形態（LR）の生成処理時間に，プロセス生成処理要求を行うために利用する遠隔手続呼出の処理オーバヘッドを加算することで算出した．また，形態（RR′）は，形態（RR）におけるプログラムに関する処理をリモートに対して行うことで実現できる．このため，形態（RR′）の生成処理時間は，形態（RR）の生成処理時間に，リモートのプログラムを利用するための遠隔手続呼出の処理オーバヘッド，すなわち，プレートの存在確認，プログラム確保，プログラム情報取得，テキスト部の読み込み，データ部の読み込み，および BSS 部の読み込みを行うために利用するそれぞれの遠隔手続呼出の処理オーバヘッドを加算することで算出した．以上を基に，算出した結果を図 9 に示す．

また，**図 11** に，プロセス削除処理時間の測定結果を示す．実測による測定対象は，表 2 の形態（LL），形態（RR），および形態（LR）の各形態でそれぞれ生成したプロセスの削除処理である．ここでは，表 4 の通番（3）に示すカーネルコールを利用して，各形態で生成したプロセスを，当該プロセスを生成したプロセスが削除を行ったときの処理時間を測定した．測定では，プロセス生成処理時間の測定と同様に，プロ

グラムの大きさとプロセス削除処理時間との関係を調べた．なお，形態（RL），ならびに形態（RR′）については，以降で述べる．

測定結果より，プロセス削除処理時間は，プログラムの大きさが同じ場合，形態（LL）が最も処理時間が短く，次いで形態（RR）と形態（LR）がほぼ同じ処理時間になる．この原因として，プロセス生成処理と同様に，遠隔手続呼出制御の利用回数が考えられる．ちなみに，形態（LL）の利用回数は 0 回，形態（RR）の利用回数は 1 回（プロセス削除），形態（LR）の利用回数は 1 回（プログラム削除）である．

次に，グラフの傾きについては，通信をともなわない形態（LL）の傾きと，通信をともなう形態（RR），ならびに形態（LR）の傾きがほぼ同じ大きさである．この原因は，各形態で遠隔手続呼出制御により処理依頼を行う際，同じ大きさのデータを転送しているためと考えられる．

また，表 2 の形態（RL），ならびに形態（RR′）におけるプロセス削除処理については，***Tender*** ver. 7.1 では処理を実現していない．そこで，遠隔手続呼出処理を個別に実測することにより，これらの処理時間を推定する．具体的には，形態（RL），ならびに形態（RR′）のプロセス削除処理時間として想定できる値を算出するために，プロセス削除処理の中で，遠隔手続呼出を利用する可能性のある処理，すなわちプロセス削除処理，ならびにプログラム削除処理について，それぞれの処理で遠隔手続呼出の処理オーバヘッドを実測により調べた．

図 12 に，プロセス削除時に利用する遠隔手続呼出の処理オーバヘッドの測定結果を示す．図 12 には，プロセス削除を行う際，遠隔手続呼出を利用する可能性のある処理について，遠隔手続呼出を利用してリモートで処理を行った場合の処理時間から，ローカル内で処理を行った場合の処理時間を引いたものを，遠隔手続呼出の処理オーバヘッドとして示している．図 12 より，プロセス削除，ならびにプログラム削除を行う場合，遠隔手続呼出の処理オーバヘッドは，プログラムの大きさに関係なく，ほぼ一定であることが分かる．

図 11 の実測結果，ならびに図 12 の結果より，表 2 の形態（RL），ならびに形態（RR′）におけるプロセス削除処理時間として想定できる値を算出した．ここで，形態（RL）におけるプロセス削除処理，ならびに形態（RR′）におけるプロセス削除処理は，形態（LR）におけるプロセス削除処理要求を，リモートに対して行うことで実現できる．このため，形態（RL），ならびに形態（RR′）の削除処理時間は，形態（LR）の削除処理時間に，プロセス削除処理要求を行うために利用する遠隔手続呼出の処理オーバヘッドを加算することで算出した．以上を基に，算出した結果を図 11 に示す．

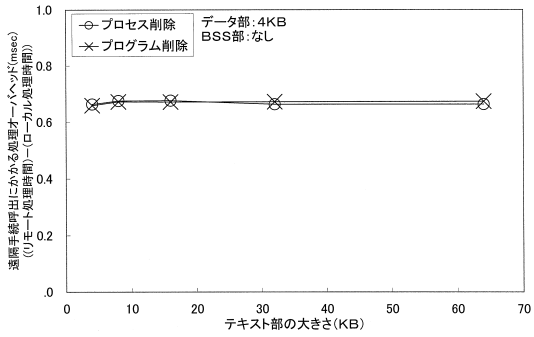

図 12 プロセス削除時に利用する遠隔手続呼出の処理オーバヘッド
Fig. 12 Overhead for remote procedure call in process deletion.

4.4.4 資源の再利用効果

Tender では，プロセスの構成要素を資源として分離し，独立化させている．したがって，プロセスを生成する際，すでに生成してある資源を構成資源として再利用することができる．また，プロセスを削除する際，そのプロセスの構成資源を解放せず，再利用するために保留しておくこともできる．このような資源の再利用効果については，ローカル計算機内に閉じている場合，すなわち，表 2 の形態（LL）におけるプロセス生成削除処理の再利用効果を，文献 2) に報告している．

文献 2) では，プロセスの構成要素の中で，再利用可能な資源をすべて再利用した場合のプロセス生成削除処理時間は，資源を再利用しない場合の処理時間と比べて，1/3 以下に短縮できることを明らかにしている．したがって，他の形態におけるプロセス生成削除処理時間は，資源の再利用を行うことにより，計算機間の通信処理時間を除けば，同程度短縮することが期待できる．

また，文献 2) では，***Tender*** と BSD/UNIX のプロセス生成処理時間を比較している．この結果より，資源の再利用効果は，OS カーネルの生成処理，ならびに Web サーバとして利用されている Apache のように，同じプログラムを実行するプロセスの生成削除が頻発する AP を利用する場合に大きいことを明らかにしている．

5. 関 連 研 究

分散環境における従来の資源操作方式の例として，

次の 2 つがある．

第 1 に，分散 OS Sprite[3] は，位置透過なネーミング機構を実現しており，ファイルシステムと入出力デバイスを透過的に利用できる．また，プロセス移動も実現している．しかし，大域的なプロセス識別子が存在しないため，位置透過なプロセス操作を行うことは難しい．具体的には，システム内の任意のプロセスが，別計算機上に存在するプロセスを，当該の計算機とは別の計算機上に移動させたり，削除したりすることが自由に行えない．

第 2 に，分散 OS V[4] は，マイクロカーネルモデルを基にして，本論文で提案した方式のように，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，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．具体的には，さらなる分割が可能な資源の生成削除を行う際に，当該資源から分割可能なすべての構成要素の生成削除を行わなくてはならない．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，資源の生成削除処理の処理負荷が大きくなってしまう．さらに，AP が細かな資源操作を行うことは難しい．逆に，資源の粒度が小さいと，各サーバ間の呼出し処理，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，本論文で提案した資源操作方式は，資源の粒度が小さい．したがって，OS は，資源の粒度が大きい場合と比べて，OS が制御する対象をより細かな単位まで資源として個別に制御できるようになる上，各資源操作に要する負荷を小さくできる．また，仮想記憶空間や実メモリのように，従来，識別や操作が制限され，プロセスの存在を前提に存在していた資源を，個別に生成し，操作することを可能にしている．さらに，資源操作を特定の処理プログラムを介するようにし，資源操作インタフェースを統一することで，AP による細かな資源操作を可能にしている．

ただし，提案方式では，資源の細分割を行っているため，処理効率の低下が懸念される．しかし，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．また，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．さらに，資源操作要求を行う際，陽にローカル指定を行うことで，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．

次に，分散環境における従来のプロセス生成の例として，次の 3 つがある．

第 1 に，UNIX を拡張した OS を利用する方法[9] がある．UNIX は，スタンドアロン環境を指向して設計され，プロセスの生成実行に fork/exec 方式を用いている．この方式は，親プロセスの走行環境を子プロセスに引き継がせることができるため，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，別計算機上に子プロセスを生成する場合，プロセス生成先の走行環境は，生成元の走行環境に依存することになる．逆に，生成元では，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，OS のカーネルレベルではなく，ユーザレベルで位置透過なプロセス生成機構を提供する方法[10] がある．この方法は，移植性は高いが，処理速度は遅くなってしまう．

第 3 に，はじめから分散環境を指向して設計した分散 OS を利用する方法[4),11),12)] がある．分散 OS V[4] では，前述したように，OS の制御対象をサーバの形で分割して管理している．しかし，資源の粒度が大きいため，プロセスを構成する細かな要素まで，個別に生成，識別，および操作することは難しい．また，分散 OS Amoeba[11] では，システムが，実行プログラムと利用プロセッサを直接指定することで，プロセス生成実行を行うため，処理効率は良い．しかし，プロセッサプールモデルに基づき，システム内の負荷情報を利用してシステム側でプロセスを配置するため，ユーザ側からは利用計算機を自由に指定できない．

一方，提案方式に基づいたプロセス生成機構では，プロセスの構成要素を細分割し，各々を資源として独立化させている．これにより，プロセスという資源の粒度を小さくすることができる．このため，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させることができる．したがって，資源の再利用により，構成資源の生成削除処理を簡略化することができるため，プロセス生成削除処理を高速化することができる．また，プロセスが利用する資源については，位置透過な資源操作方式を利用することにより，当該資源の存在場所を意識することなく利

用することができる．これにより，資源「仮想領域」を透過的に共有して利用できるため，NFS のように外部記憶装置を透過的に利用できるだけでなく，実メモリについても，分散共有メモリとして透過的に利用できるようになる．

6. おわりに

粒度が小さい資源の位置透過な操作方式を提案した．本方式では，資源を資源名と資源識別子により一意に識別できるようにし，同一形式で位置透過に扱えるようにしている．また，資源を分割する粒度を小さくすることで，各資源操作に要する負荷を小さくできる．

また，従来のプロセス資源は，その粒度が大きい．したがって，プロセスの生成，移動，および削除に要する処理負荷が大きくなるため，負荷分散の効果を低下させてしまうという問題がある．この問題への対処法として，提案した資源操作方式を基にした位置透過なプロセス生成機構について述べた．本機構では，プロセスの構成資源を分離し，独立化させている特徴を生かして，プロセスの生成，移動，および削除処理を高速化できる．

さらに，提案した資源操作方式とプロセス生成機構の実現例として，***Tender*** における実現方式を示し，資源操作処理とプロセス生成削除処理の性能を示した．その結果，資源操作処理については，陽にローカル指定した資源操作を行うことで，自計算機内における資源操作の処理効率を改善できることを示した．また，プロセス生成削除処理時間については，プロセスが利用するプログラムの大きさに比例して増加することを示した．さらに，プロセス生成削除処理における資源の再利用効果について述べた．具体的には，再利用可能な資源をすべて再利用した場合のプロセス生成削除処理時間は，計算機間の通信処理時間を除けば，再利用しない場合の処理時間と比べて，1/3 以下に短縮できることを示した．

残された課題として，計算機の場所情報表をシステム内で共有して利用する機構の実現，計算機間の通信処理における資源の再利用効果についての評価，既存方式との比較による評価，および実現した機構を利用して負荷分散を行ったときの効果についての評価がある．

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) Milojicic. D.S., Douglis. F., Paindaveine. Y., Wheeler. R. and Zhou. S.: 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) Ousterhout, J., Cherenson, A., Douglis, F., Nelson, M. and Welch, B.: The Sprite Network Operating System, *IEEE Computer*, Vol.21, No.2, pp.23–36 (1988).
4) Cheriton, D.R.: The V Distributed System, *Comm. the ACM*, Vol.31, No.3, pp.314–333 (1988).
5) 石井陽介，谷口秀夫：分散環境を指向するプロセス変身機能の提案，情報処理学会 OS 研究会報告，Vol.2002, No.13, pp.125–132 (2002).
6) 下崎　誠，谷口秀夫：***Tender*** オペレーティングシステムにおける分散共有メモリの実現と評価，情報処理学会コンピュータシステムシンポジウム論文集，Vol.99, No.16, pp.161–168 (1999).
7) 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).
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, pp.197–204 (2000).
9) 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).
10) Brent, C. and David, D.: REXEC: Decentralized, Secure Remote Execution Environment for Clusters, *Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, pp.1–14 (2000).
11) Mullender, S., Rossum, G., Tanenbaum, A., Renesse, R. and Staveren, H.: Amoeba — A Distributed Operating System fot the 1990s, *IEEE Computer*, Vol.23, No.5, pp.44–53 (1990).
12) 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).

(平成 14 年 12 月 21 日受付)
(平成 15 年 4 月 9 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．