分散環境を指向する位置透過なプロセス生成実行機構

石井陽介† 谷口秀夫††

近年，分散環境が広く利用されるようになってきている．分散環境では，複数の計算機資源を利用できる特徴を生かして，負荷分散を実現できる．負荷分散を実現するためには，位置透過なプロセス生成機構が必要になる．そこで，分散環境を指向する位置透過なプロセス生成実行機構について述べる。本機構では，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．資源は位置透過に利用でき，必要な資源を利用してプロセスを構成できる．本稿では，本機構の実現例として，*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，プロセス生成処理の性能を示す。

Location-transparent Process Creation and Execution Mechanism for Distributed Environment

YOUSUKE ISHII† and HIDEO TANIGUCHI††

In distributed system a load distribution can be realized to spread processes across all of its available computers. To realize the load distribution we need a location-transparent process creation and execution mechanism. In this paper we present this mechanism for distributed environment. In our mechanism, components of a process are divided finely, independent each other and accessed location-transparently. We describe the implementation of our mechanism on *Tender* operating system, present the processing of location-transparent process creation 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，高いCPU性能を持つものの，その利用率は低い計算機のように，CPU処理を行う余力のあ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．この負荷分散を行うためには，システム内でプロセスを静的，もしくは動的に再配置する必要がある．プロセスを静的に再配置するためには，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，プロセス移動機構が必要である．このように，分散環境の特徴を生かすためには，システム内のプロセスの管理手法が重要になる．

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，プロセスを一つの資源として扱っている．しかし，プロセスは，様々な要素により構成されているため，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，および削除処理の負荷が大きくなってしまう問題がある．そこで，本稿では，上記の問題を解決するために，分散環境を指向する位置透過なプロセス生成実行機構について述べる．本機構では，従来の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，それぞれ個別に扱えるようにする．また，各資源は，システム内で位置透過に扱えるようにする．分割に伴い，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，プロセス生成時には，必要な資源のみを利用してプロセスを構成することができる．また，プロセス移動時には，必要な資源のみを移動させるだけでよい．さらに，プロセス削除時には，構成資源を解放するかわりに，次のプロセス生成時に再利用できるよう保留しておくことができる．これらにより，プロセスの生成，移動，および削除処理の負荷を低くすることができる．また，各資源を位置透過に利用でき，利用資源の存在場所に関係なくプロセスを構成できるため，プロセスの位置透過な生成処理を容易に行うことが可能

† 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

になる．さらに，プロセスの各構成要素は独立に存在しているため，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，その上，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，資源の位置透過に利用するために，関連の計算機に対して，資源操作処理を依頼する機能は必要である．

ここで，資源を位置透過に利用する際に，その処理効率が問題となる．そこで，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陽に指定できるようにする．これにより，自計算機内の資源に対して処理を行う際に，処理効率を改善できる．また，処理の対象となる資源は，それぞれ分離し独立して存在しているため，このような指定を各資源毎に柔軟に行うことができるという利点もある．

以降では，まず，関連研究との比較について述べた後，本機構のプロセス構成法と，プロセス生成実行法について述べる．次に，本機構の実現例として，我々が開発を行っている ***Tender*** オペレーティングシステム [1] における実現方式を示す．さらに，***Tender*** における位置透過なプロセス生成処理の性能を示す．

2. 関連研究

分散環境の形態の一つとして，各計算機の OS として，UNIX を拡張した OS[2] を利用する形態がある．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．

また，他の分散環境の形態として，はじめから分散環境を指向して設計した分散 OS を利用する形態がある．分散 OS Amoeba[3] 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，処理効率はよい．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．

他には，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実行機構を提供する方法 [4] もある．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．

また，マイクロカーネルモデルを基にして，提案する機構のように，OS の制御対象を分割して管理する手法

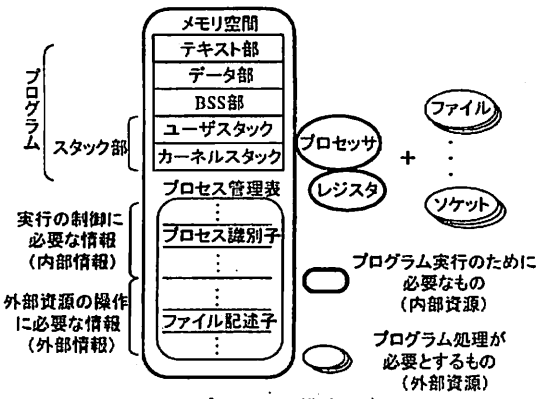

図 1　プロセスの構成要素

がある．この例として，分散 OS V[5] 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くなる．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，提案する機構では，プロセスの構成要素を資源として分割する際，資源の粒度を小さくしているため，各資源操作に要する負荷を小さくできる．また，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．さらに，プロセスの各構成要素を資源として独立化させており，プロセスは自身の構成資源について管理する必要はない．その上，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，資源の生成や削除の処理を高速化できる．ここで，各資源の管理は，計算機毎に局所化している．したがって，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．

3. 分散環境を指向したプロセス生成実行

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．例えば，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．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

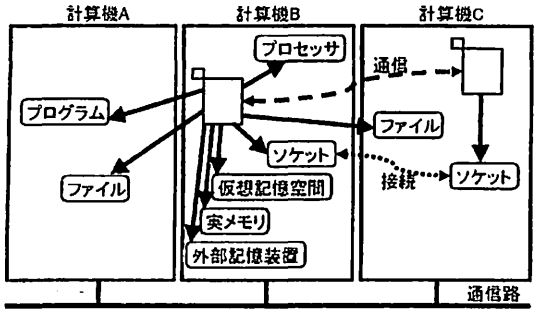

図 2 遠隔資源を利用したプロセスの構成

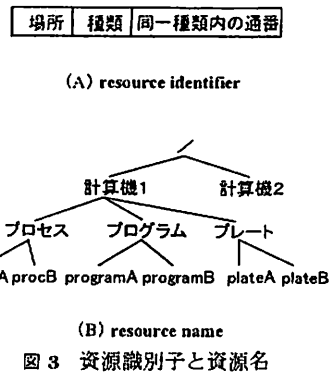

図 3 資源識別子と資源名

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．

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．プロセスを一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，プロセスを一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ので，プロセスの生成や削除に伴う処理負荷が大きくなってしまう問題がある．一方，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，低い負荷で負荷分散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 の資源を細分割する．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．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．また，資源を管理するプログラムの呼び出しは，特定の処理プログラムを介するようにする．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のみによって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，特定の処理プログラムを介して資源操作を行うことによって，応用プログラム（以降，AP と呼ぶ）による資源操作も容易に実現可能になる．

このようなプロセスの構成により，プロセスの位置透過な生成実行を容易に行うことができる．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．プロセス生成時には，これらを資源として直接指定し，操作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．

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機構を実現することが可能である．ただし，本手法では，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，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，各資源毎に陽に指定できるようにすることで，処理効率の改善を図れる．また，資源の分離独立化の特徴を生かし，資源の事前用意や保留により，資源の生成や削除に伴う処理を高速化できる．

4. *Tender* における実現方式

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

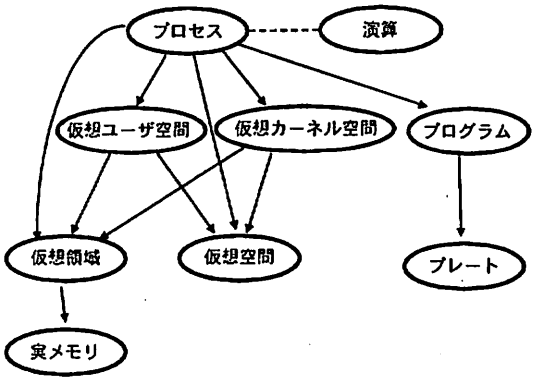

図 4　プロセスを構成する資源

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 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し，その数字や名前はシステム環境設定情報としている．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 AP 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．

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，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．

4.2　プロセスの構成

Tender では，資源の分離と独立化により，プロセスも様々な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 4 に示す．矢印は，処理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し，プロセス移動時には，当該の資源「プロセス」が，移動先計算機上に移動し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

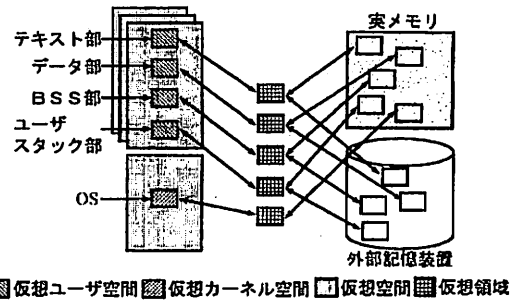

図 5　プロセスとメモリ管理関連の資源

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．

ここで，プロセスとメモリ関連資源との関係を図 5 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．両者は共に，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　プロセス生成実行法

4.3.1　特　　徴

Tender では，プロセスの生成実行に関して，次に述べる四つの特徴がある．

第一に，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，例えば，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容易に生成できる．

第二に，プロセッサの割り当て単位を，資源「演算」として資源化し，プロセスとは独立に存在させていることである．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．

第三に，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とである．これにより，資源の生成や削除に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である．

第四に，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 [6] を用いて，移

表 1　プロセスの生成形態

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

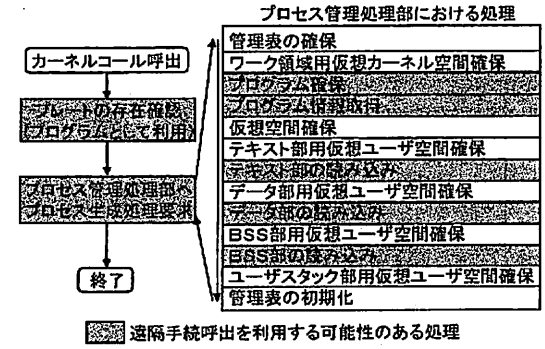

図 6　プロセス生成処理の流れ

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．

4.3.2　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 1 のように五つの形態が考えられる．ここでは，プロセス生成要求が発生した計算機をローカルとし，その他の計算機をリモートとしている．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，形態（RL）については，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′）については，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．

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 2 の（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 3 の（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 6 に示す．図において，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．ここでも，生成先がリモートの場合は，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．

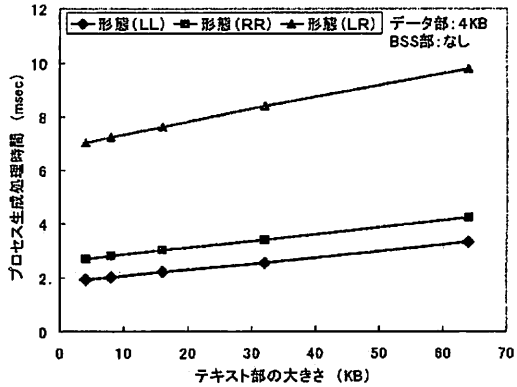

図 7　プロセス生成処理時間

4.3.3　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割り当てる必要がある．ここで，演算の生成と，割り当てのために，OS 内部で行う処理が利用するインタフェースを表 2 の（2），および（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，プロセスへ割り当てるために，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 3 の（2）に示す．表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割り当てを位置透過に実行できる．

4.4　評価と考察

プロセス生成処理を対象として，評価を行った．ここでは，*Tender* ver.7.1 において，表 3 の（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の測定結果を示す．測定対象は，表 1 の形態（LL），形態（RR），ならびに形態（LR）とした．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，他の大きさを固定（データ部:4KB，BSS部:なし）として，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．ただし，今回，資源の事前用意や保留を利用した処理の高速化は行っていない．測定環境として，計算機 (PentiumII 450MHz) を二台，通信路に Myrinet(1.28Gbps) を利用した．測定結果を図 7 に示す．

測定結果より，まず，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（プロセス管理処理部へのプロセス生成処理要求），形態（LR）の利用回数は 6 回

表 2 プロセス生成実行のために資源管理処理部が提供するインタフェース

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 を持つ演算を一つ生成し，資源名 rsc_name を付与する．
(3)	演算割り当て	attach_execution(execid, pid)	execid で指す演算を，pid で指すプロセスに割り当てる．

表 3 プロセス生成実行を行うカーネルコールインタフェース

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 を持つ演算を一つ生成し，pid で指すプロセスに生成した演算を割り当てる．

（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，今回の測定では，テストプログラムには必要ない BSS 部の読み込み処理も，遠隔手続呼出制御を利用している．これは，現版では処理の高速化を行っていないためである．

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．

さらに，測定結果より，表 1 で示している形態（RL），ならびに形態（RR′）の処理時間も推察できる．形態（RL）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）の処理時間（T_{RL}）は，形態（LR）の処理時間（T_{LR}）に，プロセス生成処理要求を行うための遠隔手続呼出処理時間を加算することで算出できる．この遠隔手続呼出処理時間は，ちょうど形態（RR）の処理時間（T_{RR}）から，形態（LL）の処理時間（T_{LL}）を引くことで算出できる．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′）の処理時間（$T_{RR'}$）は，T_{RR} に，リモートのプログラムを利用してプロセスを生成する処理時間 T_{LR} を加算することで算出できる．したがって，T_{RL} と $T_{RR'}$ は，それぞれ次の（式 1），ならびに（式 2）として表すことができる．

$$T_{RL} = T_{RR} + T_{LR} - T_{LL} \quad （式 1）$$

$$T_{RR'} = T_{RR} + T_{LR} \quad （式 2）$$

5. おわりに

分散環境を指向する位置透過なプロセス生成実行機構について述べた．本機構では，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．各資源は位置透過に利用でき，利用資源の存在場所に関係なくプロセスを構成できる．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．さらに，資源の事前用意や保留による再利用で，資源の生成削除処理を高速化できる．また，本機構の実現例として，***Tender*** における実現方式を示し，プロセス生成処理の性能を示した．その結果，位置透過なプロセス生成処理時間は，生成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．

残された課題として，UNIX プロセスの生成時間との比較，処理の高速化，および事前処理や保留処理による資源の再利用効果についての評価がある．

参考文献

1) 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).

2) 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).

3) 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).

4) Brent C. and David C.: “REXEC: Decentralized, Secure Remote Execution Environment for Clusters,” *Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, pp.1–14 (2000).

5) Cheriton D. R.: “The V Distributed System,” *Communications of the ACM*, Vol.31, No.3, pp.314–333 (1988).

6) 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).