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の提案

田端利宏† 谷口秀夫†

プロセスとは，オペレーティングシステムがプログラムの実行を制御する単位である．プロセスは，プログラムの実行時に生成され，プログラムの処理終了時に削除される．プロセスの生成処理は，メモリ空間の生成やプログラムの読込みなどの処理を必要とするため，オペレーティングシステムの処理の中でも負荷が大きい．このため，プロセスの生成処理を高速化する研究が数多くなされている．我々は，プロセスを構成する資源の再利用によるプロセスの生成と削除の処理の高速化手法を提案した．再利用可能なプロセス構成資源を，プロセスの処理終了時に保存し，プロセスの生成時に再利用することで処理を高速化する．しかし，再利用可能な資源を保存した分だけメモリ資源を消費するため，再利用可能な資源を効率良く再利用できる管理法が必要である．本論文では，プロセスを構成する資源の再利用頻度の向上，およびプロセスを構成する資源の再利用処理速度向上を目指した再利用可能な資源の管理法を提案する．

Proposal of Efficient Resource Management for Recycling Process Elements

Toshihiro Tabata† and Hideo Taniguchi†

Operating system controls a process to execute a program. A process is created for executing a program. Then the process is deleted when the program terminated. Processing of process creation needs creation of virtual address space and a read of program. The load of the processing is heavy. Therefore there are many researches for fast process creation. We proposed fast process creation mechanism by recycling process elements. Fast process creation and fast process deletion are realized by recycling process elements. However, reserved process elements consume memory resources. Therefore efficient resource management is necessary. This paper proposes efficient resource management for recycling process elements.

1. はじめに

プロセスとは，オペレーティングシステム（以降，OSと略す）がプログラムの実行を制御する単位である．OSは，プログラムを実行するために，プロセスを生成し，制御し，削除する．しかし，プロセスの生成処理は，メモリ空間の生成やプログラムの読込みなどの処理を必要とするため，OSの処理の中でも負荷が大きい．このため，プロセスの生成と削除の処理が頻発する応用プログラム（以降，APと略す）では，サービス処理性能が低下することがある．たとえば，Webサーバでは，Webサーバが要求待ちプロセスの数以上の要求を同時に受付けた場合，要求をできるだけ多く同時に処理するためにプロセスを生成する．プロセスの生成処理が短期間に集中して起こる場合，OSはプロセスの生成処理に多くのプロセッサ時間を割り当てるため，Webサーバが単位時間に処理できるトランザクション数が低下する．このため，プロセスの生成処理を高速化する研究が行われている．プロセスの生成処理を高速化する研究としては，UNIX[1)]でのsticky bitやvforkシステムコールがある．

我々は，文献2)において，***Tender***オペレーティングシステムにおける資源の独立化機構を生かした例として，プロセスを構成する資源（以降，プロセス構成資源と呼ぶ）を再利用することによりプロセスの生成と削除の処理を高速化できることを示した．具体的には，プロセス構成資源のうち，再利用可能な資源を明らかにし，その再利用可否の観点を明らかにした．また，各資源を再利用することによるプロセスの生成と削除の処理の高速化の効果を示した．しかし，再利用可能な資源を残すことは，その分だけメモリ資源を

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

消費する．このため，再利用可能なプロセス構成資源が，再利用されることなく存在し続けることは，避けなければならない．つまり，再利用可能な資源の再利用頻度を向上させる必要がある．また，プロセスの生成と削除の処理時間を短縮するには，プロセスの削除時に必要となる再利用可能な資源の登録処理，およびプロセスの生成処理で必要となる再利用可能な資源の検索処理の処理時間が，プロセス構成資源を生成し削除する処理時間よりも短い必要がある．

文献 2) では，再利用のために残した資源の管理については議論していない．また，プロセスの生成と削除の処理時間として，プロセスの生成システムコールと削除システムコールの総処理時間だけを明らかにしている．このため，登録処理，検索処理，および各資源の生成処理と削除処理の処理時間は明らかでない．そこで，本論文では，再利用可能なプロセス構成資源の管理における課題を明らかにし，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を提案する．具体的には，プロセス構成資源の再利用頻度の向上，および再利用可能な資源の登録処理と検索処理の処理時間の短縮法について述べる．最後に，提案手法を実装して評価した結果を報告する．

Tender [2]では，既存 OS が扱う資源をさらに細分化し，演算[3]やプレート[4]などの新たな資源を導入し，資源を分離し独立化している．これにより，プロセス構成資源を独立して存在させることができ，資源の再利用や事前生成によるプロセスの生成と削除の処理の高速化を実現している．これに対し，既存 OS では，プロセス構成資源を再利用する機構，およびプロセス構成資源を事前生成する機構は実現されていない．また，プロセスの生成処理を高速化する研究として sticky bit，メモリ管理の研究として On Demand Paging（ODP），および Copy on Write（CoW）がある．UNIX では，sticky bit によるテキスト部の再利用を行っている．***Tender***は，テキスト部の内容だけでなく，プロセスが使用していた仮想記憶空間も再利用できる点で異なる．ODP では，プロセス生成処理でのプログラムの読込み処理を遅延させることで，プロセスの生成処理時間を短縮している．また，UNIX では，子プロセスの生成時に，CoW を用いることにより，親プロセスと子プロセスの物理メモリを共有させ，メモリ間データ複写処理を最小限にしている．提案手法は，プロセスを構成するメモリ資源などを再利用することで，プロセスの生成処理時間を短縮しており，これらの方式とは異なる．さらに，再利用可能な資源の管理と類似した研究として，仮想記憶管理のページ管理やキャッシュメモリの管理がある[5]．文献 6) では，AP の処理結果をもとに，AP が要求するデータをプリフェッチ，またはキャッシュすることで，AP の処理を高速化している．また，文献 7) では，バッファやキャッシュを統合することにより，バッファリング処理を効率化する手法を提案している．これらの研究では，メモリ管理が提供するメモリの操作単位，たとえばページ単位で領域を管理する．一方，***Tender***でのプロセス構成資源は，物理ページだけではなく，ページテーブルやプログラム資源がある．このため，これらの研究とは扱う資源の種類が異なり，その管理手法が異なる．

また，制御を奪われたサーバが不正にアクセス制限を解除するのを防ぐ目的で，プロセス・クリーニング[8]という手法が提案されている．この手法では，リクエストを処理する度にプロセスを生成しなおすのではなく，メモリイメージを含むプロセスの状態をリクエストを受付ける前の状態に戻すことで，プロセスからクラック攻撃の影響を取り除くことを実現している．プロセスの状態をすべて複製すると，プロセス生成処理と同等もしくはそれ以上の処理負荷を要するため，この手法では，copy-on-write 機能を利用してメモリイメージの保存と復元の処理に要するメモリのコピー量を減らしている．また，この手法は，プロセスごとにその状態を保存し復元する．このため，この手法では，保存した状態を他のプロセスの復元処理に利用できない．一方，本論文で提案する手法では，再利用のために残したメモリ資源を多くのプロセスで再利用することができ，OS 全体のプロセスの生成と削除処理の高速化が期待できる．たとえば，プロセス・クリーニングという手法では，プログラムが同じ大きさでもプログラムが異なる場合，保存した内容を利用できないが，本提案手法では，プログラムの種類が異なってもメモリ資源を再利用できる．さらに，プロセス・クリーニングという手法を利用するために，プロセスの状態を保存するシステムコールと復元するシステムコールを発行するように AP を変更する必要がある．一方，本提案手法では，OS レベルで自動的に資源を再利用する機構を実現しているため，AP を改版する必要がない．

2. 資源再利用によるプロセスの生成と削除の高速化手法

2.1 基 本 機 構

プロセスの生成処理の高速化は，再利用可能なプロセス構成資源を再利用し，プロセス構成資源を生成する処理を短縮することにより実現する．プロセスの生

成処理では，OSが再利用可能な資源を検索し，再利用可能な資源がある場合，プロセス構成資源を生成する処理を再利用処理に置き換える．また，プロセスの削除処理の高速化は，プロセスの削除時に再利用可能なプロセス構成資源を，再利用可能な資源を管理する管理表（以降，再利用可能資源管理表と呼ぶ）に登録し，実際に削除しないことにより，プロセス構成資源の削除に要する処理時間を短縮することにより実現する．

プロセス構成資源を再利用するためには，プロセス構成資源が個別に単独で存在できる必要がある．しかし，既存 OS の多くは，プロセス構成資源の情報がプロセス管理表に存在し，プロセス構成資源の存在はプロセスの存在を前提とする．このため，プロセス構成資源が個別に単独で存在できないという問題がある．そこで，文献 2) は，プロセス構成資源を細かく分類し，プロセス構成資源が独立して存在できることを実現した***Tender***オペレーティングシステム[2]上で，プロセスの生成と削除の処理の高速化機構を提案した．

以降，2 章では，文献 2) で提案した***Tender***の資源の分離と独立化機構，プロセス構成資源，および資源再利用によるプロセスの生成と削除の処理の高速化手法について説明する．

2.2 *Tender*オペレーティングシステム

Tenderでは，OS が操作する対象を資源として，分離し独立化した．資源には，**図 1** に示す資源識別子と資源名を付与し，資源操作のインタフェースを統一している．さらに，各資源を操作するプログラム部品（以降，資源管理処理部と呼ぶ）を資源ごとに分離し，共有プログラムを排除した．また，各資源の管理情報も資源ごとに分離し，各資源の管理表の間のポインタを禁止した．このように，資源の分離と独立化を行うことで，資源の事前生成や保留により，資源の作成や削除をともなう処理を高速化している．さらに，プログラムを部品化できるため，機能の追加や変更が容易である．

OS の資源の分離と独立化の例として，プロセスの生成と削除の処理におけるプロセス構成資源の処理の呼び出し関係を**図 2** に示す．図 2 の矢印は，資源間の処理の呼び出し関係を表している．プロセス構成資源を分離し独立化した結果，プロセス資源とプロセス構成資源の関係が明確になり，プロセス構成資源は，プロセス資源の存在に関係なく存在することが可能となった．たとえば，UNIX の場合，プロセスが削除すると，プロセスが利用していたメモリ空間を解放しなければならないため，メモリ空間の再利用はできない．これに対し，***Tender***では，プロセスを削除させるときに当該プロセスが利用していたメモリ空間を残すことができる．

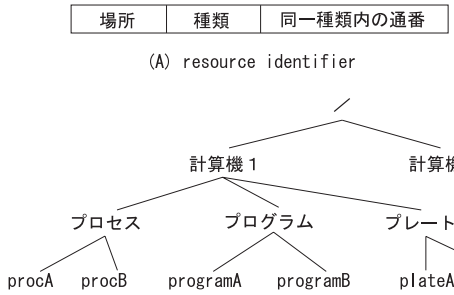

図 1 資源識別子と資源名
Fig. 1 Resource identifier and resource name.

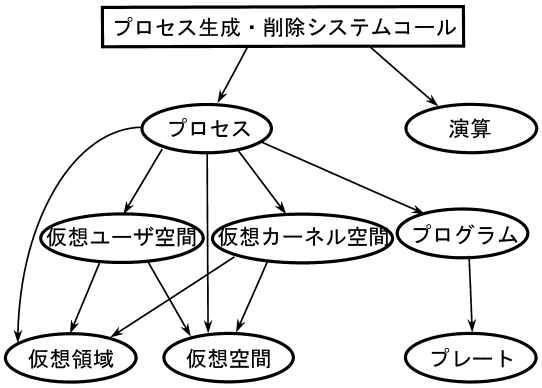

図 2 プロセスの生成と削除時の資源の呼び出し関係
Fig. 2 Relation of resources on process creation and disappearance.

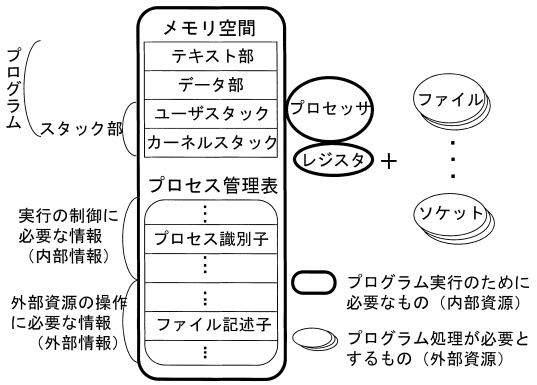

図 3 プロセスの構成要素
Fig. 3 Elements of process.

2.3 プロセス構成資源

プロセスの構成要素を**図 3** 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（これを内部資源と呼ぶ），

表 1 プロセスの作成と削除のインタフェース
Table 1 Interface of process creation and disappearance.

形式	引数の内容	機能
open_proc (plateid, arg, vmid)	plateid：プロセスとして起動するプログラムを識別する識別子 arg：プロセスへの引数 vmid：プロセスを生成する仮想空間の識別子（0 のときは新規仮想空間）	plateid で指定されたプログラムをプロセスとして生成する．vmid ≠ 0 であれば，vmid で指定された仮想空間上にプロセスを生成する．
close_proc (pid, rflag)	pid：削除するプロセスの識別子 rflag：再利用のため残存させる資源を指示するフラグ	rflag で指定された資源を保存し，プロセスを削除する．

表 2 プロセス構成資源の再利用可否の観点
Table 2 List of recyclable resources.

通番	再利用可能な資源	再利用可否の観点
1	ワーク領域用仮想カーネル空間	（つねに再利用）
2	内容を利用するテキスト部用仮想ユーザ空間	プログラムの内容とアドレス位置
3	内容を利用しない仮想ユーザ空間（テキスト部，データ部，BSS 部，ユーザスタック部用）	アドレス位置と大きさ
4	仮想空間	仮想空間の内容
5	仮想領域（テキスト部，データ部，BSS 部，ユーザスタック部用）	大きさ
6	プログラム	プログラムの内容

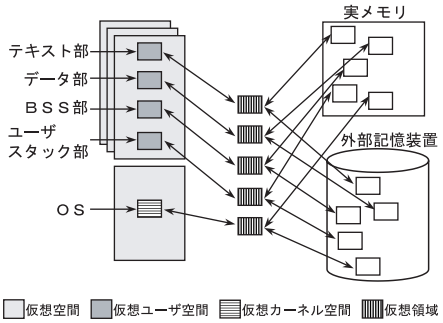

図 4 プロセス構成資源間の関係
Fig. 4 The relation of resources in the *Tender* process.

プログラムの処理が必要とするもの（これを外部資源と呼ぶ）がある．内部資源には，メモリ空間，プロセッサ，レジスタなどがあり，外部資源には，ファイルやソケットなどがある．***Tender***のプロセス資源は，内部資源のうち，メモリ空間，およびプロセス管理表から構成される．***Tender***のプロセスの構成要素は，図 2 に示したように，資源として分割され，それぞれ独立して管理される．図 2 で示した「プログラム」とは，実行プログラムについて，テキスト部とデータ部の大きさと先頭アドレス，および開始アドレスの情報を持つ資源であり，実行プログラムの形式を隠蔽する．また，プログラムは，プレートを用いて実行プログラムを外部記憶装置上ではなくメモリ上に格納している．プレートとは，メモリ上のデータに永続性を持たせた資源であり，既存 OS のファイルに相当する資源である．

プロセス構成資源間の関係を**図 4** に示す．仮想空間とは，特定のアドレス領域を持つ仮想的な空間であり，その実体はページディレクトリとページテーブルからなるアドレス変換表である．仮想領域とは，メモリイメージを仮想化した領域である．仮想領域の実体は，実メモリ，または外部記憶装置上に存在する．仮想ユーザ空間は，仮想領域をユーザ空間用の仮想空間に貼り付けることで作成される．「貼り付ける」とは，仮想アドレスを実アドレスに対応付けることである．具体的には，当該の仮想アドレスに対応するアドレス変換表のエントリに，仮想領域の管理表が保持する実アドレスまたは外部記憶装置のアドレスを設定する．仮想ユーザ空間を削除するには，仮想領域を仮想空間から剥がす処理を行う．「剥がす」とは，仮想アドレスと実アドレスの対応付けをなくすことである．仮想カーネル空間は，仮想領域を OS 用の仮想空間に貼り付けることにより作成される．プロセスのテキスト部，初期値を持つ変数や文字列の集合部分であるデータ部，初期値を持たない変数や文字列の集合部分である BSS 部，およびユーザスタック部は，仮想ユーザ空間に読み込まれた状態で仮想空間上に存在する．

2.4 提供インタフェース

表 1 に，プロセスの生成と削除の処理のインタフェースを示し，説明する．open_proc は，指定されたプログラムを識別する識別子 plateid から，プロセスを仮想空間 vmid 上に生成する．プロセス構成資源を再利用できない場合，OS は**図 5** に示した処理を行う．再利用可能な資源が存在する場合，OS は当該資源を生成する代わりに資源を再利用する．close_proc は，プロセス pid を削除する．プロセスの削除処理では，フラグ rflag で指定したプロセス構成資源の情報を，再利用可能資源管理表に登録する．具体的には，rflag は

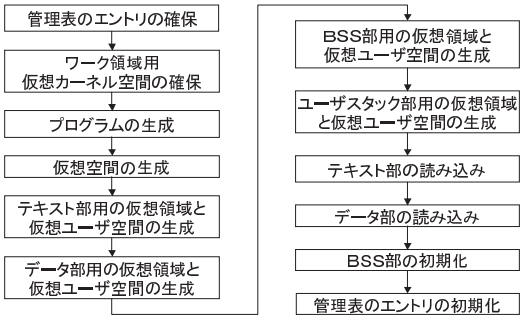

図 5 プロセスの生成処理の流れ（資源再利用なし）
Fig. 5 Flow of process creation without recycling.

unsigned int 型の変数で，下位から 6 ビットの各ビットが**表 2** で示した 6 種類の再利用可能な資源に対応している．再利用可能な資源に対応するビットが 1 のとき，当該資源の情報を再利用可能資源管理表に登録する．

2.5 資源の再利用可否の観点

プロセス構成資源を再利用するための観点を表 2 に示し，説明する．ワーク領域用仮想カーネル空間は，引数 arg を新規プロセスのユーザスタック領域に複写するための一時的なワーク領域である．生成するプロセスに渡すことのできる引数は 4 KB 未満であるため，このワーク領域の大きさは 4 KB である．このため，再利用可能資源管理表に登録されているワーク領域用の仮想カーネル空間を無条件に再利用できる．

仮想ユーザ空間は，仮想空間上のメモリ空間を提供する資源である．仮想ユーザ空間には，仮想ユーザ空間のメモリ空間に加え，さらにメモリ空間上のデータも再利用できるもの（以降，内容を利用する仮想ユーザ空間と呼ぶ）もある．一方で，メモリ空間として再利用できるものの，そのメモリ空間上のデータを再利用できない仮想ユーザ空間（以降，内容を利用しない仮想ユーザ空間と呼ぶ）もある．内容を利用するテキスト部用の仮想ユーザ空間とは，テキスト部用のメモリ空間に加え，そのメモリ空間上のテキスト部のデータも再利用できる仮想ユーザ空間のことである．これは，テキスト部は書込み不可の領域のため，メモリ空間上のテキスト部のデータは不変であることによる．したがって，テキスト部に読み込んでいるプログラムの内容が同じで，かつその仮想ユーザ空間の先頭アドレス位置が一致する場合に内容を利用するテキスト部用の仮想ユーザ空間を再利用できる．

内容を利用しない仮想ユーザ空間は，その先頭アドレス位置と大きさの情報により，再利用可否の判断ができ，テキスト部，データ部，BSS 部，およびユーザスタック部について再利用できる．仮想ユーザ空間が存在する仮想空間も同時に再利用する．

仮想空間は，一定の大きさの仮想アドレス空間を提供するアドレス変換表である．したがって，仮想ユーザ空間が存在しない仮想空間は，つねにプロセスの生成時に再利用できる．

仮想領域は，テキスト部，データ部，BSS 部，およびユーザスタック部について再利用可能であり，特定の大きさを持つ領域であるため，大きさの情報のみで再利用可否の判断ができる．

プログラムは，プログラムの内容が同じ場合，再利用できる．

2.6 資源の再利用機構の利用法

資源の再利用機構の利用法には，以下の 3 つがある．

(1) プロセスの削除時に残した資源を再利用する．
(2) AP の動きを解析し[9]，AP の処理に合わせてプロセス構成資源を生成しておき，プロセスの生成処理を高速化する．
(3) プロセッサアイドル時間を利用し，カーネルがプロセス構成資源を事前生成し保持しておくことで，プロセスの生成処理を高速化する．

3. 再利用可能な資源の管理法

本章では，再利用可能な資源を効率的に再利用するために，プログラムの実行頻度に着目した資源管理法を提案する．

3.1 考 え 方

再利用可能な資源を存在させることは，メモリ資源を消費するため，再利用可能な資源を効率よく再利用する必要がある．効率のよい資源の再利用とは，プロセスの生成時に高い確率でプロセス構成資源を再利用できることである．また，再利用のために残したメモリ資源の量に対するプロセス生成と削除の処理時間短縮効果が大きいことである．

まず，再利用可能な資源を，大きく 2 つに分類する．1 つは，特定のプログラムでのみ再利用できる資源（以降，プログラム依存資源と呼ぶ）である．もう 1 つは，特定のプログラムでなくても，条件が合えば再利用できる資源（以降，プログラム非依存資源と呼ぶ）である．プログラム依存資源は，特定のプログラムでしか再利用できないが，再利用できたときの処理時間の短縮効果が大きい．一方，プログラム非依存資源は，条件が合うプログラムで再利用できるが，プログラム依存資源に比べて，再利用できたときの処理時間短縮効果は小さい．

プログラムには，実行頻度の高いものと低いものが

表 3 再利用のために残す資源の形態
Table 3 Patterns of reserved resources for recycling.

形態	残す資源	再利用可否の観点
プログラムの内容を意識する形態	内容を利用するテキスト部用仮想ユーザ空間，内容を利用しない仮想ユーザ空間（データ部，BSS 部，ユーザスタック部），仮想空間，プログラム	プログラムの内容
プログラムの内容を意識しない形態	仮想空間，仮想領域（テキスト部，データ部，BSS 部，ユーザスタック部）	仮想空間の内容（仮想空間），大きさ（仮想領域）
つねに再利用	ワーク領域用仮想カーネル空間	無条件で再利用可

あり，実行頻度の高いプログラムを実行しているプロセスは，プログラム依存形式で残すと，同じプログラムが再実行されるときに再利用できる可能性が高い．たとえば，Web サーバなどのサーバプログラムは，繰り返し処理を実行する．このとき，プロセスの生成処理をともなうことが多いため，サーバプログラムについてはプロセスをプログラム依存資源として残すことで，プロセスの生成と削除処理を高速化でき，かつ資源の再利用回数も多いことが期待できる．一方，実行頻度が低いプログラムは，再実行される可能性が低い．このため，実行頻度が低いプログラムを実行しているプロセスをプログラム非依存形式で残すことによって，他プログラムのプロセス生成時にプロセス構成資源が再利用されることが期待できる．

さらに，資源の再利用機構を実現するには，プロセスの生成処理では再利用可能な資源の検索処理，プロセスの削除処理では再利用可能な資源の登録処理について検討する必要がある．これは，資源を再利用することによりプロセスの生成と削除の処理を高速化するには，資源の検索処理と登録処理の処理時間を短くすることも必要なためである．

3.2 再利用頻度の向上手法

Tenderでは，プログラム依存資源としては，プログラム資源および内容を利用するテキスト部用の仮想ユーザ空間がある．また，プログラム非依存資源として，内容を利用しない仮想ユーザ空間，仮想ユーザ空間が存在しない仮想空間，および仮想領域がある．

実行頻度の高いプログラムについては，再実行されることが期待できるため，プログラム依存資源を含めた形で当該プロセスのすべての構成資源を残すこととする（以降，プログラム内容を意識する形態と呼ぶ）．また，実行頻度が低いプログラムについては，プログラム非依存資源の形で当該プロセスのすべての構成資源を残すこととする（以降，プログラム内容を意識しない形態と呼ぶ）．再利用のために残す資源の形態を**表 3** に示し，資源の形態について述べる．

Tenderのプロセス構成資源は，大きく次の 3 つに分類できる．

(1) メモリ資源
(2) プログラム
(3) ワーク領域

メモリ資源については，プログラム依存資源とプログラム非依存資源の 2 つがある．プログラムの内容を意識する形態では，同一プログラムについてのみ再利用するため，テキスト部，データ部，BSS 部のアドレス位置と大きさは同一と考えることができる．このため，仮想空間，仮想ユーザ空間（テキスト部用，データ部用，BSS 部用，ユーザスタック部用）を再利用できる形態で残す．テキスト部については，その内容も再利用する．***Tender***では，ユーザスタック部のアドレス位置は，プロセス識別子の値で決定され[10]，プロセス識別子が異なればアドレス位置も異なる．このため，ユーザスタック部のアドレス位置が異なる場合，ユーザスタック部用の仮想ユーザ空間を削除し，生成し直すこととする．このとき，ユーザスタック部用の仮想領域を再利用する．これらの資源は，プロセスの生成時にまとめて再利用できるため，一括して管理する．

一方，プログラムの内容を意識しない形態の場合，プログラムごとにテキスト部，データ部，および BSS 部のアドレス位置が異なることが多いため，仮想ユーザ空間ではなく，大きさのみで再利用可否の判断できる仮想領域を残すこととする．このために，プロセスの削除処理では，仮想ユーザ空間を削除するために，仮想領域を仮想空間から剥がす処理が必要となる．ただし，他のプロセスとテキスト部用の仮想領域を共有している場合，他プロセスがテキスト部用の仮想領域を利用しているため，当該仮想領域を再利用対象とはしない．また，仮想ユーザ空間が存在しない仮想空間も残すこととする．各部の仮想領域，および仮想空間は，まとめて再利用できない場合があるため，個別に管理する．

プログラム資源は，プログラムの内容を意識するため，実行頻度が高いプログラムの場合に残すこととする．

ワーク領域は，プロセスの生成処理でつねに必要と

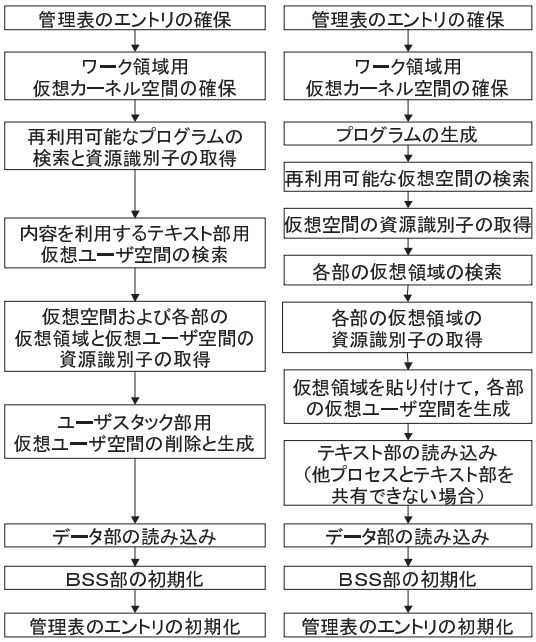

図 6 プロセスの生成処理の流れ（資源再利用あり）
Fig. 6 Flow of process creation with recycling.

なるため，つねに再利用のために残し再利用する．

各形態について，資源をすべて再利用できたときのプロセスの生成処理の流れを**図 6** に示す．プログラムの内容を意識する形態における資源の生成と削除の処理は，ユーザスタック部用の仮想ユーザ空間の削除と生成のみである．ただし，ユーザスタックのアドレス位置が同一の場合，仮想ユーザ空間の削除と生成処理は行わない．図 6（2）のプログラムの内容を意識しない形態における資源の生成と削除の処理は，プログラム資源の生成と各部の仮想ユーザ空間の生成である．

3.3 再利用可能資源の登録処理と検索処理の高速化

再利用可能資源管理表の操作としては，プロセスの削除処理における情報の登録処理，およびプロセスの生成処理における検索処理と削除処理がある．このため，情報の登録処理と検索後の削除処理を O(1) で行えるリストを用いて，再利用可能資源管理表を管理する．また，リストを用いることにより，管理する資源の増加に合わせて，動的に管理表のメモリ領域を確保できる利点が得られる．

一方で，再利用可能な資源の検索時間が課題となる．無条件で再利用の可否を判断できる資源については，再利用可能な資源の有無を確認するだけであり，検索時間は O(1) である．具体的には，ワーク領域用仮想カーネル空間と仮想空間がある．検索時間が O(n) と

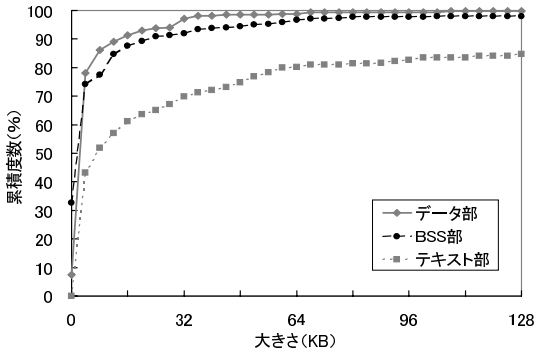

図 7 各部の大きさの累積度数（テキスト部，データ部，BSS 部）
Fig. 7 Cumulative frequency of region size (text region, data region and bss region).

なる資源として，プログラムの内容を意識する形態，および仮想領域がある．プログラムの内容を意識する形態の場合，プログラムの内容が一致する資源を検索する．プログラムデータは，既存 OS のファイルに相当する資源であるプレート[4]上に存在する．そこで，プログラムの内容を識別するために，プレートの資源識別子を利用する．OS は，プレートの資源識別子の通番部分を，プレート資源を生成した順番で決定する．このため，プログラムの内容を意識する資源をプレート識別子の通番部分の下位数ビットをハッシュ関数のキーとしてチェイン法で管理する．これにより，均等な確率で管理する情報をリストに振り分けることができる．また，仮想領域の大きさは，ページサイズ（4 KB）の整数倍である．このため，仮想領域の大きさ（ページ数）をハッシュ関数のキーとしてチェイン法で管理する．ハッシュ関数を決定するために，プログラムのテキスト部，データ部，および BSS 部の大きさの偏りを調査した．BSD/OS ver.3.1 について/bin，/usr/bin，/usr/local/bin に存在するプログラム（総数 338 個）のテキスト部，データ部，および BSS 部の大きさを調査し，各部の大きさの累積度数を求めた結果を**図 7** に示す．

図 7 から次のことが分かる．プログラムのテキスト部の大きさは，約 80%のプログラムが 64 KB 以下である．データ部については 98.8%，BSS 部については 96.7%のプログラムが 64 KB 以下の大きさである．このことから，多くの仮想領域は，64 KB 以下の大きさであるといえる．また，***Tender***では，ユーザスタック部の大きさは 64 KB で固定である．このため，多くの仮想領域は，64 KB 以下の大きさであるといえる．したがって，64 KB 以下の仮想領域については，ハッシュ関数を用いてページ数ごとのリストに振り分けることで，80%以上の仮想領域を O(1) で検索でき

る．一方，64 KB より大きい仮想領域については，1つのリストで管理することとした．このため，64 KB より大きい仮想領域の検索処理は，O(n) となる．しかし，同一プログラムから生成したプロセス間ではテキスト部を共有するため，テキスト部で利用される仮想領域はプログラムごとに 1 つである．このため，テキスト部用の仮想領域は，データ部と BSS 部の仮想領域に比べ，再利用される頻度が少ないため，64 KB より大きい仮想領域が再利用可能となる可能性は小さいといえる．

なお，本節では，ハッシュ関数を利用する閾値となる仮想領域の大きさを決定するために，静的なプログラムの解析を利用し，***Tender***では 64 KB を閾値とした．しかし，目的とするシステムにより，実行するプログラムの種類やその実行頻度が異なるため，64 KB という値が最もよいとは限らない．したがって，目的とするシステムに合わせて，ハッシュ関数の閾値を最適なものに変更する必要がある．

4. 評　　価

4.1 測定内容と測定環境

プロセス構成するメモリ資源の生成処理と削除処理の処理時間を測定し，メモリ資源の特徴を示す．次に，登録処理と検索処理の例として，登録処理が O(1) で，検索処理が O(1) または O(n) である仮想領域について登録処理と検索処理を評価した．最後に，プログラムの内容を意識する形態とプログラムの内容を意識しない形態の特徴を明らかにするため，プロセスの生成処理と削除処理を評価した．

測定は，Pentium III 750 MHz の PC/AT 互換機上で行い，プロセッサのカウンタを利用して測定した．

4.2 メモリ資源の生成と削除の処理時間

仮想空間，仮想領域，および仮想ユーザ空間の生成と削除の処理時間を**表 4**，**図 8**，および**図 9** に示す．

仮想空間は，ページディレクトリとページテーブルからなる．仮想空間は，生成時にページディレクトリ部分のみ生成される．ページテーブル部分は，仮想空間上に仮想ユーザ空間を生成するときに，仮想ユーザ空間を生成するアドレス範囲に対応する部分が生成される．ただし，すでに当該アドレス範囲を含むページテーブルが存在する場合は，そのページテーブルをそのまま利用する．このため，仮想ユーザ空間の生成処理では，生成するアドレス範囲のページテーブルの有無により，処理時間が大きく異なる．なお，ページテーブルを生成する際には，仮想カーネル空間が生成される．

表 4 仮想空間の生成と削除の処理時間
Table 4 Processing time of virtual memory space creation and deletion.

処理内容	処理時間（マイクロ秒）
仮想空間の生成処理	359.6
仮想空間の削除処理（ページディレクトリなし）	31.5
仮想空間の削除処理（ページディレクトリあり）	56.1

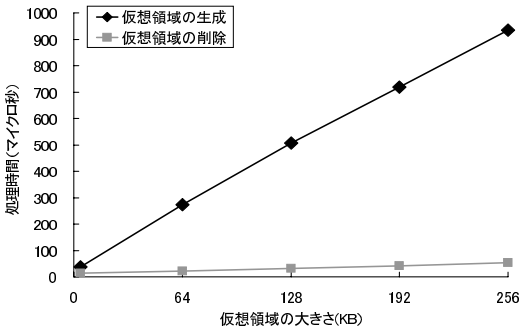

図 8 仮想領域の生成と削除の処理時間
Fig. 8 Processing time of virtual region creation and deletion.

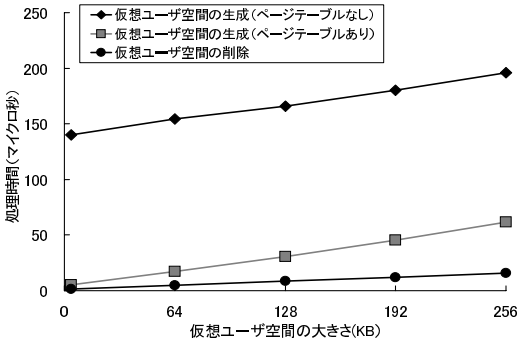

図 9 仮想ユーザ空間の生成と削除の処理時間
Fig. 9 Processing time of virtual user space creation and deletion.

図 9 では，ページテーブルの有無により，仮想ユーザ空間の生成の処理時間が大きく異なる．この処理時間の差は，ページテーブル用の仮想カーネル空間の生成とその初期化処理に要する時間である．このことから，ページテーブルを生成するコストが大きいことが分かる．また，表 4 から，ページテーブルが存在する場合の仮想空間の削除処理時間が長いことが分かる．これは，ページテーブルが，仮想ユーザ空間の削除処理では削除されず，仮想空間の削除処理で削除されるためである．以上のことから，ページテーブルを持つ仮想空間を再利用すると，ページディレクトリに加えページテーブルを再利用でき，仮想ユーザ空間の生成

表 5 仮想領域の登録と検索の処理時間
Table 5 Processing time of virtual region registration and search.

処理内容	処理時間（マイクロ秒）
仮想領域の登録処理	0.15
仮想領域の検索処理（64 KB 以下）	0.09

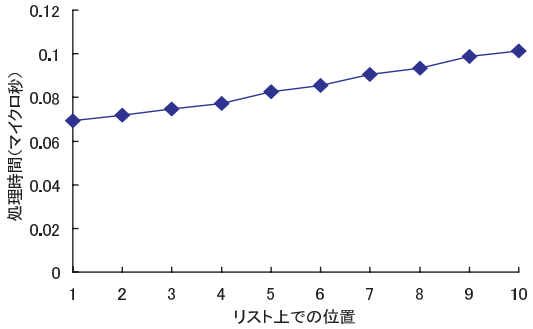

図 10 仮想領域の検索処理時間（64 KB より大）
Fig. 10 Relation between position on a list and processing time of search.

処理時間を短縮できるため，仮想空間の再利用の効果は大きいといえる．

仮想領域は，確保する領域の大きさに比例して生成処理時間が増加する．これは，仮想領域の生成時に，仮想領域の大きさに見合った管理表を仮想カーネル空間上に生成することによる．さらに，実メモリ資源を生成し，その情報を管理表に格納するためでもある．実メモリ資源は，4 KB で 1 資源であるため，仮想領域の大きさに比例して確保する実メモリの資源数が増加する．仮想領域の削除処理時間は，実メモリ資源を削除する処理をともなうため，仮想領域の大きさに比例してわずかに処理時間が増加する．

4.3 再利用可能な資源の登録処理と検索処理の評価

仮想領域の登録処理と検索処理の処理時間を**表 5** と**図 10** に示す．表 5 から，仮想領域の登録処理はわずか 0.15 マイクロ秒であることが分かる．これは，図 8 の仮想領域の削除時間（約 20 マイクロ秒）の 1%以下である．

次に，表 5 から，仮想領域の検索処理が O(1) の場合（仮想領域の大きさが 64 KB 以下の場合），検索処理は 0.1 マイクロ秒以下であり，仮想領域の生成処理（39 マイクロ秒以上）の 1%以下であることが分かる．図 10 は，再利用可能な資源がリスト上で 1 番目から 10 番目の位置で見つかった場合の検索処理時間を示している．10 番目の位置にある情報を検索する処理時間は 0.12 マイクロ秒未満であり，100 番目につながっていても，0.5 マイクロ秒程度である．したがって，仮想領域の生成処理時間（64 KB で約 270 マイクロ秒）に比べると 1%以下であり非常に小さい．以上の結果から，再利用可能な資源の登録処理，および検索処理の処理時間は，資源の生成と削除の処理時間に比べ，十分短いといえる．

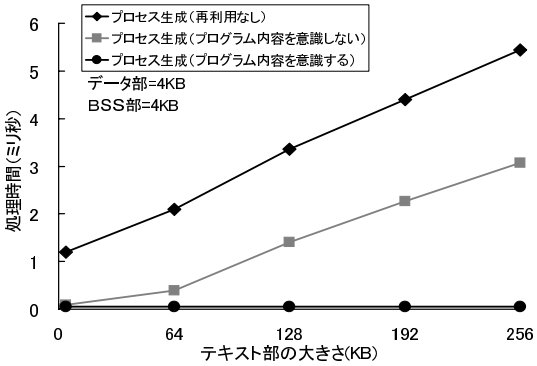

図 11 プロセスの生成処理時間（テキスト部サイズ可変）
Fig. 11 Processing time of process creation (text region size was changed).

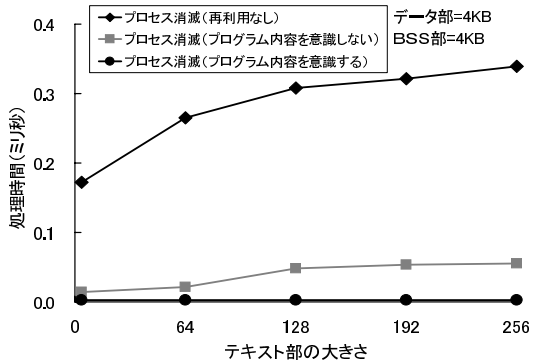

図 12 プロセスの削除処理時間（テキスト部サイズ可変）
Fig. 12 Processing time of process deletion (text region size was changed).

4.4 資源の形態による再利用効果と特徴

4.4.1 測定内容と条件

再利用する資源の形態による再利用効果と特徴を明らかにするため，表 1 に示した資源「プロセス」の資源管理処理部が提供するプロセスの生成と削除のインタフェースの呼び出しから終了までの処理時間を測定した．また，資源を再利用しない場合，プログラムの内容を意識しない場合，およびプログラムの内容を意識する場合について測定した．**図 11**，および**図 12** は，テキスト部の大きさのみを可変とした場合の測定結果である．**図 13**，および**図 14** は，データ部の大きさのみを可変とした場合の測定結果である．BSS 部を可変にした場合については，データ部と同様の結果

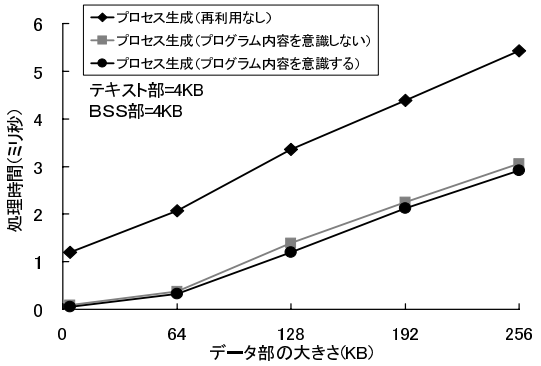

図 13 プロセスの生成処理時間（データ部サイズ可変）
Fig. 13 Processing time of process creation (data region size was changed).

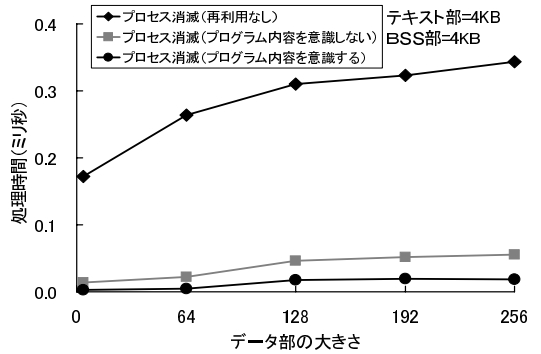

図 14 プロセスの削除処理時間（データ部サイズ可変）
Fig. 14 Processing time of process deletion (data region size was changed).

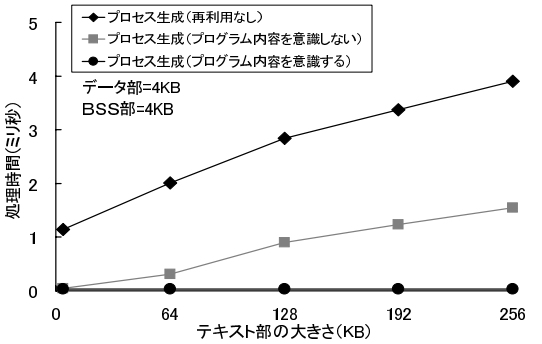

図 15 プロセスの生成処理時間（テキスト部サイズ可変，メモリ間データ複写の処理時間を除く）
Fig. 15 Processing time of process creation without memory copy (text region size was changed).

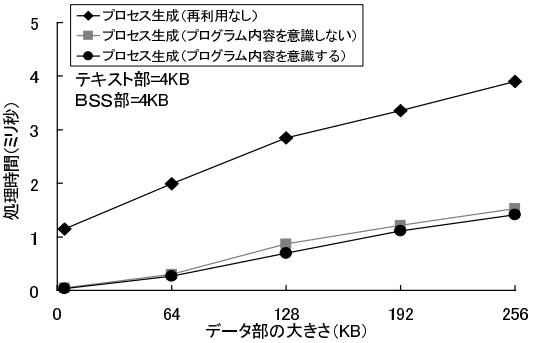

図 16 プロセスの生成処理時間（データ部サイズ可変，メモリ間データ複写の処理時間を除く）
Fig. 16 Processing time of process creation without memory copy (data region size was changed).

が得られるため省略した．なお，プレート[4]を利用してプログラムをメモリ上に常駐させ，ディスク I/O が発生しない条件で測定した．

4.4.2 プロセスの生成処理

プロセスの生成処理時間の測定結果である図 11 と図 13 から以下のことが分かる．テキスト部の大きさ可変の場合（図 11），プログラムの内容を意識する形態では，プロセスの生成処理時間が一定（約 0.06 ミリ秒）であり，資源再利用の効果が大きい．これは，テキスト部の内容を再利用するため，テキスト部の内容を仮想ユーザ空間に複写する必要がないためである．その他の場合は，プロセスの生成処理時間がテキスト部やデータ部の大きさに比例する．いずれの場合においても，プログラムの内容を意識する形態とプログラムの内容を意識しない形態は，再利用なしの場合に比べて 7 ミリ秒以上処理時間が短い．たとえば，図 13 のプログラム内容を意識しない形態では，データ部の大きさが 4 KB の場合で 1 ミリ秒（総処理時間の約 92%），データ部の大きさ 256 KB の場合で 2.4 ミリ秒（総処理時間の約 44%）処理時間を短縮できる．また，プログラムの内容を意識する形態は，プログラムの内容を意識しない形態に比べて処理時間が短い．図 13 におけるプログラム内容を意識する場合とプログラム内容を意識しない場合の差は，データ部の大きさが 4 KB のとき 0.04 ミリ秒で，データ部の大きさが 256 KB のとき 0.14 ミリ秒であり，プロセスの大きさとともに増大する．

現在の ***Tender*** では，ODP を実装していないため，プロセスの生成時にメモリ上にプログラムの内容を複写する．ODP を実現すれば，プロセス生成時にプログラム内容のメモリ間データ複写処理を行う必要はない．また，メモリ間複写の処理時間を除くことで，メモリ資源の再利用の効果を明確にすることができる．そこで，プロセスの生成処理時間から，プログラム内容のメモリ間複写の処理時間を除いたものを**図 15** と**図 16** に示す．

図 15 から，プログラム内容を意識する場合，プロ

セスの生成処理時間が一定の処理時間（約 0.03 ミリ秒）であることが分かる．これは，プログラム内容を意識する形態では，プロセスのメモリ資源をすべて再利用しているため，処理時間へのプロセスの大きさによる影響がないためである．

以上のことから，テキスト部が大きいプログラムを繰り返し実行する場合に，プロセスの生成と削除の処理の高速化効果が特に大きいといえる．なお，ODP を実現している場合，データ部が大きいプログラムに対しても効果が特に大きいといえる．

4.4.3 プロセスの削除処理

プロセスの削除処理の測定結果である図 12 と図 14 から以下のことが分かる．プログラム内容を意識する場合，処理時間は，テキスト部の大きさとデータ部の大きさに関係なく，0.02 ミリ秒以下であることが分かる．プログラムの内容を意識しない場合，処理時間は，プロセスの大きさとともに増加するものの，資源を再利用しない場合に比べて，10 分の 1 程度である．これらのことから，資源の再利用によるプロセスの削除処理時間の短縮効果は大きいといえる．

4.5 プログラム実行ログをもとにした評価

4.5.1 実行回数と各部の大きさの関係の評価

実際のプログラムの実行頻度に基づき提案手法を評価するために，Web サーバや利用者のメール処理や文書作成などの実行環境として利用している計算機上で acct コマンドを利用してプログラムの実行ログを 24 時間分取得した．実行ログから，実行されたプログラムの各部の大きさの累積度数を算出した結果を**図 17** に示す．プログラムのテキスト部の大きさは，37.6%のプログラムが 64 KB 以下である．データ部については，97.5%，BSS 部については 95.4%が 64 KB の大きさである．このことから，データ部と BSS 部については，その大きさが図 7 での静的解析の結果と同様に小さいことが分かる．しかし，テキスト部については，図 7 での評価に比べて，64 KB より大きい仮想領域の割合が多いことが分かる．一方，3.3 節でも述べたように，テキスト部用の仮想領域はプロセス間で共有されるため，データ部や BSS 部に比べ，再利用される数は少ない．このため，64 KB より大きいテキスト部用の仮想領域が多くても，資源の管理方法の違いによる影響は少ないと考えられる．

次に，実行ログから求めた実行回数が上位のプログラムを**表 6** に示す．実行されたプログラム数は 23,589 回であり，実行回数のうち全体の 58.4%を上位 5 つのプログラムが占めていることが分かる．3 章では，実行頻度が高いプログラムは，プログラム内容を意識する形態で残す方がよいことを述べた．たとえば，実行回数の多い上位のプログラムについて，プログラムの内容を意識する形態で残す方法が考えられる．

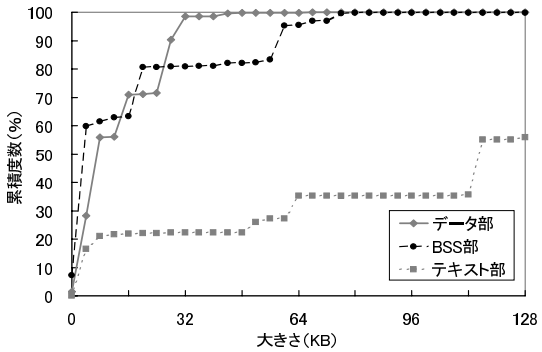

図 17 実行されたプログラムの各部の大きさの累積度数（テキスト部，データ部，BSS 部）
Fig. 17 The cumulative frequency of a region size (text region, data region and bss region).

表 6 実行回数が上位のプログラム
Table 6 The information of programs, that is the top 8 of execution times.

名前	回数	割合（%）	Text	Data	BSS
nmbd	3,827	16.2	323584	28672	18380
sh	3,750	15.9	118784	8192	3348
sendmail	2,624	11.1	503808	16384	59552
perl	1,847	7.8	413696	32768	2332
popper	1,746	7.4	65536	8192	3772
ls	743	3.1	6864	304	228
expr	734	3.1	3876	2256	24
date	727	3.1	3204	336	12
合計	23,589	100.0	—	—	—

4.5.2 ベンチマークプログラムによる評価

実際のプログラムの実行頻度における提案手法の効果を明らかにするために，実行ログに合わせて，プロセスを生成し削除するプログラムを作成した．4.5.1 項の実行ログは，プログラムの実行回数が多いため，4.5.1 項の実行ログの最初のプログラムから 500 番目のプログラムまで，同様に 600 番目まで，同様に 700 番目まで，同様に 800 番目まで，同様に 900 番目まで，同様に 1000 番目までについて，計 6 つの場合について評価した．なお，評価に用いた実行ログの最初のプログラムから 500 番目までのプログラムついては，そのプログラムの実行割合が表 6 と同様であることを確認した．このように実行個数を 500 個から 1,000 個とした理由は，後の考察に示すように，評価する各場合の特徴を如実に示すには十分な個数であるからである．また，資源を再利用できるか否かについて，プログラムを実行する順序の影響が大きいため，実システ

ムでのプログラムの実行ログをもとに評価した．

評価では，プロセスの生成と削除処理の処理時間短縮効果を明らかにするため，ベンチマークプログラムの処理時間を測定した．また，再利用のためのメモリ消費量に対する処理時間短縮効果を明らかにするため，再利用可能な資源のメモリ使用量も測定した．

評価は，プログラム内容を意識する形態ですべて再利用した場合（以降，すべて意識と呼ぶ），プログラム内容を意識する形態とプログラム内容を意識しない形態を併用した場合（以降，部分的に意識と呼ぶ），およびプログラム内容を意識しない形態ですべて再利用した場合（以降，意識しないと呼ぶ）で測定した．部分的に意識では，表 6 の実行頻度上位 5 つまでのプログラムについてのみプログラム内容を意識する形態で資源を残した．なお，ベンチマークプログラム実行前には，再利用可能な資源は存在しない条件で測定した．

処理時間の測定結果を**図 18** に，メモリ使用量の測定結果を**図 19** に示す．図 18 から，意識しない場合は，他の場合に比べ，2 倍以上処理時間が長いことが分かる．一方で，メモリ使用量は最も少ない．次に，プログラムの実行回数 500 回と 600 回の場合，部分的に意識の場合は，すべて意識の場合よりも処理時間が短いことが分かる．これは，初めてプログラムを実行する場合，すべて意識の場合では資源を再利用できないのに対し，部分的に意識の場合ではプログラム内容を意識しない資源である仮想領域や仮想空間を再利用できることがあるためである．しかし，プログラムの実行個数が増えると一度実行したプログラムを再実行することが増加するため，プログラム内容を意識する形態の資源の再利用が有効に働き，すべて意識の場合の処理時間が部分的に意識よりも短くなっている．しかし，すべて意識のメモリ消費量は，部分的に意識の 1.5 倍である．

以上のことから，部分的に意識は，メモリ消費量が意識しない場合に比べわずかに多いものの，処理時間は意識しない場合の半分以下である．また，部分的に意識は，すべて意識と処理時間の差がわずかあるが，メモリ消費量はすべて意識の約 3 分の 2 である．つまり，実行頻度の高いプログラムをプログラム内容を意識する形態で残す効果が大きいといえる．また，実行頻度の低いプログラムをプログラム内容を意識しない形態で残すことで，資源を異なるプログラム間で再利用でき，再利用に必要となるメモリ消費量を削減できることが分かる．さらに，プログラム内容を意識しない形態で残された資源は，初めて実行されるプログラムに再利用され，有効に働いているといえる．

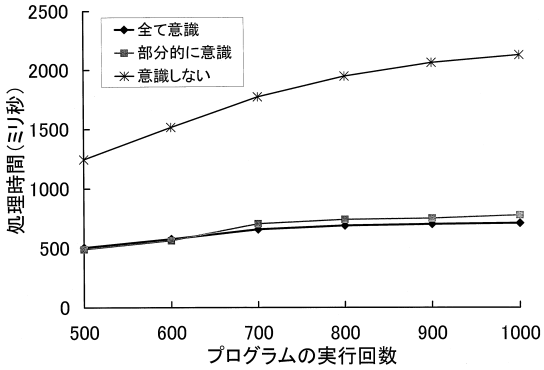

図 18 ベンチマークプログラムにおける処理時間
Fig. 18 The processing time of the benchmark program.

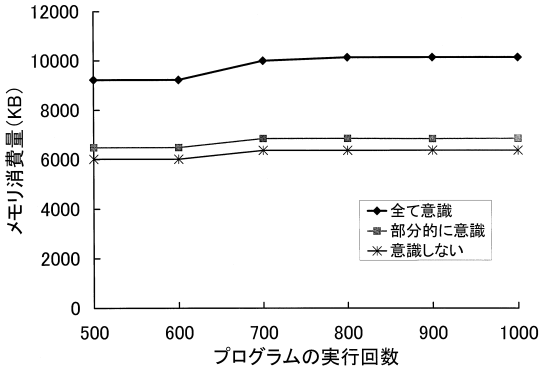

図 19 ベンチマークプログラムにおけるメモリ消費量
Fig. 19 The memory consumption of the benchmark program.

したがって，プログラム実行頻度に合わせて，再利用可能な資源を残す形態を設定することで，メモリ消費量を抑えることができ，かつ資源再利用によるプロセスの生成と削除時間の短縮効果を十分に生かすことができる．ただし，実行されるプログラムの種類が決まっており，それらのプログラムをプログラムを意識する形態で再利用するのに十分なメモリがある場合は，プログラム内容を意識する形態ですべてのプログラムを残すほうが良いと推察できる．

5. ま と め

再利用可能なプロセス構成資源の管理法について述べた．資源の再利用頻度を向上させるには，実行頻度の高いプログラムについてはプログラムの内容を意識する形態で資源を残し，実行頻度の低いプログラムについてはプログラムの内容を意識しない形態で残すことが重要である．また，プロセスの生成と削除の処理を高速化するには，再利用可能な資源の登録処理と検索処理を高速化する必要がある．このために，ハッ

シュ関数を利用して資源を管理する方法を示した．これにより，登録処理と削除処理は O(1) で実現できる．また，検索処理については，プロセス構成資源の特徴に合わせてチェイン法も利用する方式を示した．

実装し評価した結果，再利用可能な資源の登録処理と検索処理の処理時間は，プロセス構成メモリ資源を新たに生成する処理時間に比べ，十分に短いことを示した．具体的には，プログラムの内容を意識する形態では，プロセスの生成処理時間がテキスト部の大きさに関係なく一定（0.5 ミリ秒）であることを示した．また，プログラムの内容を意識する形態とプログラムの内容を意識しない形態のいずれにおいても，処理時間を 7 ミリ秒以上短縮できることを示した．たとえば，プログラム内容を意識しない形態では，データ部の大きさが 4 KB の場合で 7.0 ミリ秒（総処理時間の 92.5%），データ部の大きさ 256 KB の場合で 17.1 ミリ秒（総処理時間の 71%）処理時間を短縮できる．さらに，プロセスの削除処理時間を 10 分の 1 以下に短縮できることも示した．これらのことから，プロセス構成資源を再利用する効果は大きいといえる．

実際のプログラムの実行頻度における提案手法の効果を明らかにするために，実システムのプログラムの実行ログに合わせて，プロセスを生成し削除するプログラムを作成した．評価結果から，プログラム実行頻度に合わせて，再利用可能な資源を残す形態を設定することで，メモリ消費量を抑えることができ，かつ資源再利用によるプロセスの生成と削除時間の短縮効果を十分に生かすことができることを明らかにした．

残された課題として，再利用可能な資源の削除に関する検討，実 AP での評価，およびプロセス構成資源の事前生成機構の検討がある．

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) Quarterman, J.S., Silberschatz, A. and Peterson, J.L.: 4.2BSD and 4.3BSD as Examples of the UNIX System, *ACM Computing Surveys*, Vol.17, No.4, pp.379–418 (1985).
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：***Tender*** オペレーティングシステムにおける資源「演算」を用いたサービス処理時間の保証，情報処理学会論文誌，Vol.41, No.6, pp.1745–1754 (2000).
4) 稲本慎司，谷口秀夫：***Tender*** においてメモリ上のデータを永続化する資源「プレート」の復元機構，情報処理学会第 63 回全国大会講演論文集（分冊 1），pp.85–86 (2001).
5) Braunstein, A., Riley, M. and Wilkes, J.: Improving the Efficiency of UNIX File Buffer Caches, *SOSP: 12th ACM Symposium on Operating Systems Principles*, pp.71–82 (1989).
6) Patterson, R., Gibson, G., Ginting, E., Stodolsky, D. and Zelenka, J.: Informed Prefetching and Caching, *SOSP'95: 15th ACM Symposium on Operating Systems Principles*, pp.79–95 (1995).
7) Pai, V.S., Druschel, P. and Zwaenepoel, W.: IO-Lite: A Unified I/O Buffering and Caching System, *OSDI'99: USENIX Association 3rd Symposium*, pp.15–28 (1999).
8) 光来健一，千葉 滋：サーバのアクセス制限を安全に変更するための機構，情報処理学会論文誌，Vol.42, No.6, pp.1492–1502 (2001).
9) Suranauwarat, S. and Taniguchi, H.: The Design, Implementation and Initial Evaluation of an Advanced Knowledge-based Process Scheduler, *ACM Operating Systems Review*, Vol.35, No.4, pp.61–81 (2001).
10) 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，情報処理学会研究報告，Vol.98, No.33, pp.87–94 (1998).

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

田端 利宏（正会員）

平成 10 年九州大学工学部情報工学科卒業．平成 12 年同大学大学院システム情報科学研究科修士課程修了．平成 14 年同大学院システム情報科学府博士後期課程修了．平成 13 年日本学術振興会特別研究員．平成 14 年九州大学大学院システム情報科学研究院助手．博士（工学）．オペレーティングシステムに興味を持つ．電子情報通信学会会員．

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．