

プロセス構成要素を再利用した プロセスの生成と消滅の高速化

田端 利宏 谷口 秀夫

九州大学大学院システム情報科学研究科

プロセスの各構成要素を再利用することによるプロセスの生成と消滅の高速化の有効性について報告する。*Tender*では、プロセスの構成要素を資源として分離し独立化している。そのため、プロセスとは独立してプロセスを構成する資源が存在できる。このことを利用し、事前に資源を作りおきしたり、すでに存在する資源を再利用したりすることにより、プロセス生成をの高速化できる。また、プロセス消滅時には、プロセスが確保していた資源をプロセスと一緒に消滅させずに、再利用のために残すことにより、プロセスの消滅を高速化できる。この資源再利用の機能を *Tender* に実装し実測評価した結果、プロセスに関連するすべての資源を再利用することによりプロセスの生成と消滅の時間を 15 ミリ秒以上高速化できる。

Fast Process Creation and Disappearance by Recycling Process Elements

Toshihiro TABATA and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering, Kyushu University

We suggest the mechanism of fast process creation and disappearance by recycling process elements. Process elements of *Tender* are separated and independent. Therefore process elements are able to exist without process existence. Fast process creation is realized by creating resources beforehand or by recycling resources. And fast process disappearance is realized by remaining resources of process. We implemented this mechanism of resource recycle on *Tender* and evaluated it. As the result, the time of process creation and disappearance is fast more than 15 millisecond.

1 はじめに

近年、プロセッサの性能は向上し、複雑なソフトウェアを処理することが可能になった。また、あらゆる分野に計算機は普及し、高速な通信ネットワークで結ばれ計算機ネットワークを形成するようになった。そのため、オペレーティングシステム(以降、OSと略す)への要求は多様化し、OSの機能は高度化している。OSの多機能化や高度化を実現するために、その開発では、OS機能に重点がおかれることが多く、プログラム構造は軽視されがちであり複雑化している。そこで、我々は、プログラム構造を重視して機能を実現する *Tender*^[1] (The ENduring operating

system for Distributed EnviRonment)を開発している。

本稿では、従来のOSのプロセス構造の問題点について述べ、その問題点を解決するために、プロセスの各構成要素を資源として分離し独立化した *Tender* オペレーティングシステムのプロセス構造について述べる。資源を独立化したことにより、プロセスを構成する資源がプロセスとは個別に存在することが可能となった。このことを利用した資源を再利用して、プロセスの生成と消滅の高速化について述べ、さらに、実測評価について述べる。

2 プロセス構造の問題点と対処

2.1 プロセス管理構造

プロセスとは、プログラムをメモリ上に読み出し、実行する時、OSがその動作を制御する基本単位である。プロセス管理は、プロセスを生成し、制御し、終了させることができる機能を持つ。プロセスは、生成され、動作を制御され、終了する。プロセスの状態は、プロセス管理表により制御される。

以下では、プロセスの構成要素について説明する。図1は、プロセスの構成要素を図示したものである。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)、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。プログラムは、テキスト部、データ部、スタック部(ユーザスタック、カーネルスタック)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と、初期値を持たない変数の集合部分からなる。後者はBSS部と呼ばれている。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、ユーザスタックはプロセスがユーザモードで走行する時に利用し、カーネルスタックはプロセスが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(内部情報)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(外部情報)に分類できる。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。

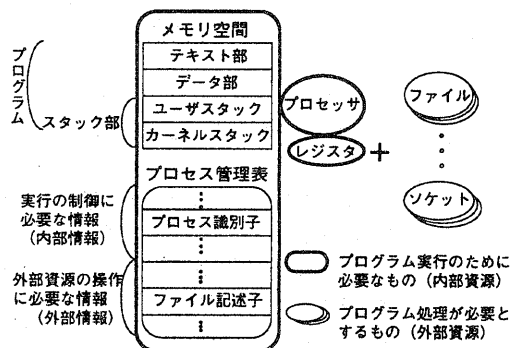


図1 プロセスの構成要素

2.2 従来のOSのプロセス構造の問題点

従来のOSのプロセス構造には大きく二つの問題がある。一つは、プロセスの管理情報をさまざまな

モジュールが共有し一つの管理表として保持している点である。このため、プロセスの管理情報をさまざまなモジュールが操作するため、動作の把握が困難である。また、プロセスの管理情報を明確に分類できないため、構造の理解が困難である。さらに、機能の追加や変更がさまざまなモジュールへ影響を与えるため、その改版が困難である。

もう一つは、プロセスの各構成要素の存在はプロセスの存在を前提としている点である。これは、プロセスの各構成要素の管理情報が、一つの管理表に格納されているため、プロセスが存在しなければ管理情報が存在しないために起こる。このため、プロセスの削除の際に、別の新たなプロセスが再利用できるプロセス構成要素まで解放してしまう。例えば、プロセスの存在していた仮想メモリ空間は、プロセス消滅時にプロセスと一緒に解放することになる。また、プロセスの存在しない仮想メモリ空間を作ったりすることはできない。プロセスの存在とは関係なく独立してプロセス構成要素を作ることができないため、プロセス構成要素を事前に作りおくことができない。したがって、プロセスの消滅時に再利用可能なプロセス構成要素を解放し、プロセスの生成時には、プロセス構成要素を再利用したり、作りおきしたプロセス構成要素を利用できないため、プロセスの生成と消滅のオーバーヘッドが大きくなっている。

これらの問題を解決するため、Tenderでは、プロセスの各構成要素の関係を明確化し「資源」として分離し独立化させ、これに伴い資源毎に管理表を分離した^[2]。

2.3 Tenderのプロセス管理構造の特徴

Tenderでは、プロセスの各構成要素の関係を明確化し資源として分離し独立化させた。資源の分離と独立化の様子を、図2に示す。ここで、資源とは、「一つの処理を行なう場合、その操作で処理が完結し、他の処理を必要としないもの」と定義する。

従来のプロセス管理では、各資源管理部は他の資源管理部を意識することなく、直接他の資源を操作していた。しかし、資源を分離し独立化させたことにより、他の資源を直接操作することができなくなったが、各資源間の関係を調べ、各資源管理部間の操作の依頼関係を明確化させることができた。他の資源管理部に対して操作を依頼する関係を主従関係と呼ぶ。

また、各資源管理部は、他の資源管理部を意識することなく管理情報を操作する。そのため、管理情報の関係は、明確にされていなかった。そこで、プロセスの管理情報を明確化した上で、各資源毎に管理情報を共有しないように分離した。各資源管理部は、資源を管理する管理表と資源を操作するための制御表を持ち、それらを利用して、資源を管理し制御する。このことにより、プロセスの管理情報をさまざまな資源が共有することはなくなり、プロセスの管理情報共有の問題を解決できた。

また、資源を分離し独立化させたため、プロセスの存在を前提とせずに、各資源が独立して存在できる。このことにより、プロセスの各構成要素の存在はプロセスの存在を前提としているという問題を解決できた。

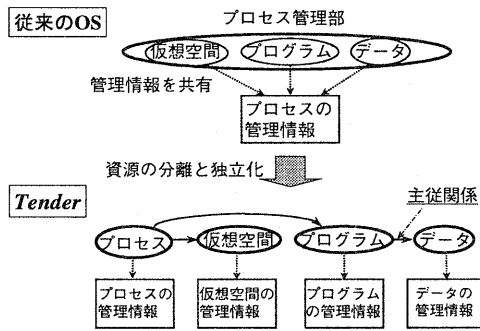


図2 資源の分離と独立化

3 プロセスの生成と消滅の高速化

3.1 プロセスとメモリ管理関連の資源

プロセスとメモリ管理関連の資源の関係を図3に示す。

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（以降、データ部と呼ぶ）、BSS部、ユーザスタック部は、仮想ユーザ空間^[3]に読み込まれた状態で、仮想空間^[3]上に存在する。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る。仮想ユーザ空間とは、メモリイメージを仮想化した領域である仮想領域^[3]をユーザ空間用の仮想空間に貼り付けることで作成できる。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。「貼り付ける」とは、仮想アドレスを実アドレスに対応付けすることである。仮想カーネル空間^[3]とは、仮想領域をOSの存在する仮想空間に貼り付けることに

より作成される。

プロセス生成時のメモリ関連の処理を具体的に説明する。

- (1) プロセス管理部は、プロセス生成時に、一つの仮想空間を確保する。
- (2) 次に、テキスト部、データ部、BSS部、ユーザスタック部のそれぞれについて、以下の処理を行なう。
 - (a) 仮想領域を作成する。
 - (b) その仮想領域を、仮想空間に貼り付ける。貼り付けることにより、仮想ユーザ空間が作成され、その空間にアクセスできるようになる。
 - (c) テキスト部やデータ部用の仮想ユーザ空間の場合には、その内容を仮想ユーザ空間に読み込む。

(3) 最後に、プロセス管理表の初期化を行なう。以上の処理をすることにより、プロセスは実行可能な状態になる。

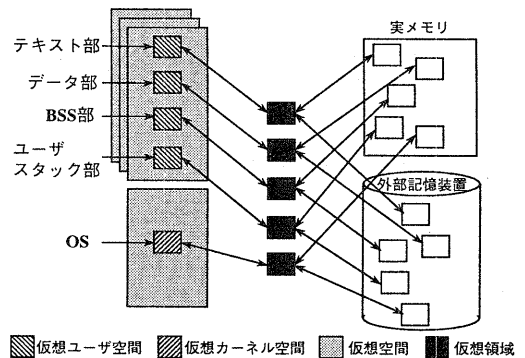


図3 プロセスとメモリ管理関連の資源

3.2 プロセスの生成と消滅の高速化の概要

プロセスの生成と消滅時の各資源管理部の呼びだし関係を図4に示し以下に資源再利用の仕組みについて説明する。

プロセスの生成時に、プロセス管理部は、プログラム、仮想空間、仮想ユーザ空間、仮想領域、仮想カーネル空間の各管理部に対して、処理を依頼する。各資源管理部に処理を依頼する時には、確保したい資源の情報を与え、各資源管理部から、確保した資源に関する情報を受け取る。したがって、プロセスの消滅時に、再利用可能な資源に関する情報をプロセス管理が保存しておけば、プロセスの生成時に要

表 1 再利用可能資源の一覧

通番	再利用可能資源	再利用可否の観点
1	ワーク領域用仮想カーネル空間	(常に再利用)
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

求する資源と同じ条件のものが、保存しておいた再利用可能資源に関する情報の中にあるか調べ、同じ条件のものがあればその情報を使って資源を再利用できる。

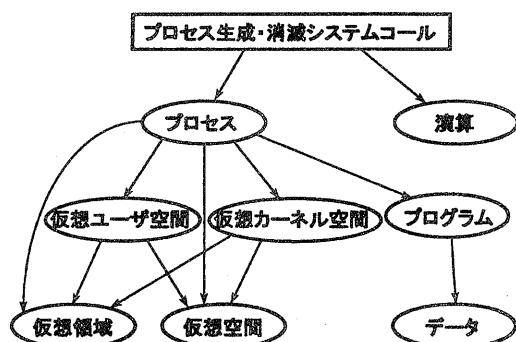


図 4 プロセスの生成と消滅時の資源管理部の呼びだし関係

これにより、各資源管理部に資源の生成を依頼することなく、資源を利用することができる。

つまり、プロセスの生成の高速化は、プロセスとは独立して存在する資源を事前生成または再利用することで、実現できる。また、プロセスの消滅の高速化は、再利用可能な資源を解放せずに保存することで、実現できる。

3.3 再利用可能資源

*Tender*においてプロセスを構成する資源のうち再利用可能な資源と再利用の可否の観点を表 1 に示し以下に説明する。

ワーク領域用仮想カーネル空間とは、プロセスの生成時に利用する一時的な領域である。プロセスの生成時に必要とするワーク領域の大きさは常に 4KB と一定なので、OS を起動して最初にプロセスを生成する時に作ったワーク領域を再利用のために保存し

ておき、そのワーク領域の情報をプロセス管理が持つておく。そして、それ以降のプロセス生成処理でその情報を利用することにより、この資源を再利用できる。この部分は、プロセスの生成時には、必ず必要な資源であり、かつその大きさは小さく固定 (4KB) なので、常に再利用した方が良く考えられる。

テキスト部の仮想ユーザ空間は、書き込み不可なのでその内容を利用する場合と利用しない場合に分類できる。内容を利用する場合は、プログラムの内容と同じであれば、テキスト部の仮想ユーザ空間の内容も同じであるため、プログラムの内容とアドレス位置により、再利用可否の判断ができる。

内容を利用しない仮想ユーザ空間場合は、テキスト部、データ部、BSS 部、ユーザスタック部について再利用可能であり、アドレス位置とその大きさにより、再利用可否の判断ができる。

仮想空間は、仮想空間に仮想ユーザ空間が貼り付いていなければ、仮想ユーザ空間を再利用しない時でも再利用できる。また、仮想ユーザ空間を再利用する時は、仮想ユーザ空間の貼り付いている仮想空間も再利用することになる。

仮想領域は、テキスト部、データ部、BSS 部、ユーザスタック部について再利用可能であり、ある大きさを持つメモリ空間なので、大きさ情報のみで再利用可否の判断ができる。

プログラムは、そのプログラムの内容が同じかどうかを調べることで再利用可否の判断ができる。

3.4 資源再利用インタフェース

プロセスの作成と削除のインタフェースを表 2 に示し、以下に説明する。再利用可能資源は、各資源の再利用を個別に自由に選択できるために、各資源をビットに対応させたフラグ (rflag) で選択する。プロセスの生成処理では、資源を再利用する方が必ず

表2 プロセスの作成と削除のインタフェース

通番	形式	引数の内容
1	open_proc(dataid, arg, vmid)	dataid : プログラムを識別する識別子 arg : プログラムへの引数 vmid : プロセスを生成する仮想空間の識別子 (0 の時は新規空間)
2	close_proc(rflag)	rflag : 再利用のため残存させる資源を指示するフラグ

速いと推測できるので、そのフラグを見るのは、プロセスの消滅時の処理のみとする。

プロセスを消滅させる際に、フラグで選択された資源の情報をテーブルに登録し、その資源を保存する。この管理は、プロセス管理が行なう。

プロセス生成時の処理では、テーブルに登録されている資源があるかどうかを調べ、あればテーブルから再利用可能な資源を検索し、再利用可能な資源がテーブルに存在すれば、その資源を再利用してプロセスを生成する。

3.5 実現と評価

3.5.1 測定環境

プロセスの生成と消滅を高速化できる資源再利用機能を *TenderVer.3* に実装した。

測定は、プロセッサ i486DX2(66MHz) メモリ 32MB の計算機上で、*TenderVer.3* を走行させ、プロセスの生成と消滅の処理を 1000 回繰り返して行ない、処理の平均時間を算出した。プロセスの大きさは、BSS 部 4KB、ユーザスタック 8KB とし、テキスト部とデータ部のサイズを変えて測定した。資源再利用の効果を把握しやすくするため、ディスク I/O は発生しない環境とした。また、時間は 1ms 周期で割り込むタイマを用いて測定した。

3.5.2 すべての資源を再利用した場合の効果

すべての資源を再利用した場合とそうでない場合について、データ部サイズとプロセス生成消滅時間の関係を図5に示す。テキスト部サイズは 4KB 固定で一定である。

図5より、プロセス生成消滅時間は、すべての資源を再利用することにより、資源を再利用しない場合に比べて、15ms 以上高速化できる。また、図5の二つの場合の傾きの差分より、データ部サイズに対し約 0.35ms/4KB の時間短縮効果があることがわかる。

3.5.3 ワーク領域用仮想カーネル空間と仮想空間の再利用効果

ワーク領域用仮想カーネル空間と仮想領域と仮想空間の再利用効果についての測定結果を表3に示す。

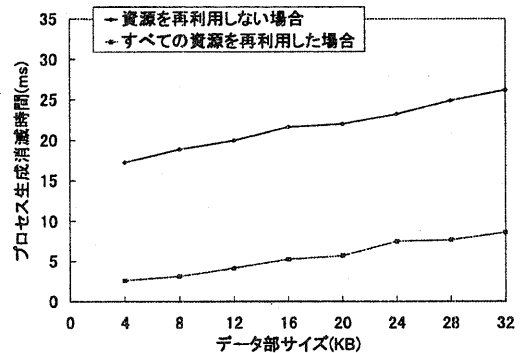


図5 資源再利用の効果 (テキスト部 4KB 固定)

この測定でのプロセスのサイズは、テキスト部、データ部が 4KB と固定で一定である。

表3より、資源を再利用しない場合 (通番 1) と、ワーク領域用仮想カーネル空間のみを再利用した場合 (通番 2) のプロセス生成消滅時間の差は、1.45ms である。つまり、ワーク領域用仮想カーネル空間を再利用することにより、プロセス生成消滅を 1.45ms 高速化できることがわかる。

次に、仮想カーネル空間の作成にかかる時間の内訳について考察する。仮想カーネル空間の作成は、まず仮想領域を作成し、その仮想領域をカーネル空間用の仮想空間に貼り付けることにより実現できる。したがって、仮想カーネル空間の作成は、仮想領域の作成と仮想領域の仮想空間への貼り付けの処理に時間を要することがわかる。そこで、仮想領域のみを再利用した場合 (通番 3) と、仮想領域とワーク領域用仮想カーネル空間を再利用した場合 (通番 4) を比べる。表3の通番3と通番4から、後者の方が 0.25ms 早く処理が終ることがわかる。この二つの場合の処理の違いは、ワーク領域用仮想カーネル空間用の仮想領域を仮想空間に貼り付けるかどうかという違いしかないので、この時間の差は 4KB の仮想領域を仮想空間に貼り付ける処理時間ということがわかる。したがって、1.20ms の時間が、4KB の仮想領域の作成時間ということがわかる。

表 3 ワーク領域用仮想カーネル空間と仮想領域と仮想空間の再利用効果

通番	再利用資源	プロセス生成消滅時間 (ms)
1	資源を再利用しない場合	17.22
2	ワーク領域用仮想カーネル空間	15.77
3	仮想領域 (ワーク領域用仮想カーネル空間、テキスト部、データ部、BSS 部、ユーザスタック部用)	11.45
4	ワーク領域用仮想カーネル空間 仮想領域 (テキスト部、データ部、BSS 部、ユーザスタック部用)	11.20
5	ワーク領域用仮想カーネル空間 仮想空間	8.86

仮想空間の再利用効果を、表 3 から推察する。仮想空間には、大きさや位置などの情報はなく、プロセスの生成時に必要な仮想空間は一つであるため、再利用効果は一定である。表 3 の通番 2 と通番 5 の差から、仮想空間の再利用により 6.91ms の高速化ができることがわかる。

3.5.4 内容を利用するテキスト部用仮想ユーザ空間の再利用効果

すべての資源を再利用した場合と内容を利用するテキスト部用仮想ユーザ空間以外の資源を再利用した場合について、テキスト部サイズとプロセス生成消滅時間の関係を、図 6 に示す。

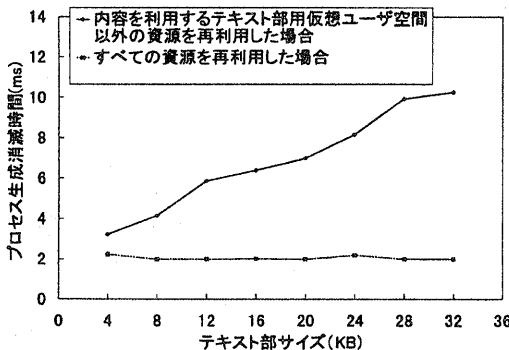


図 6 内容を利用するテキスト部用仮想ユーザ空間の再利用効果 (データ部 4KB 固定)

図 6 より、内容を利用するテキスト部用仮想ユーザ空間を再利用することで、テキスト部サイズに関係なくプロセス生成消滅時間をほぼ一定にできることがわかる。これは、テキスト部用の仮想ユーザ空間の内容を利用することにより、テキスト部用の仮想ユーザ空間にテキスト部の内容を読み込むため処理が必要なくなるためである。今回の測定では、ディスク I/O は発生しない環境にしているため、二つの

場合のグラフの差は、テキスト部の内容をメモリ間で複写する時間を表している。

したがって、内容を利用するテキスト部用仮想ユーザ空間の再利用は、テキスト部サイズの大きなプロセスを生成する場合に効果が大きいといえる。

3.5.5 内容を利用しない仮想ユーザ空間の再利用効果

内容を利用しない仮想ユーザ空間を再利用した場合と資源を再利用しない場合について、データ部サイズとプロセス生成消滅時間の関係を、図 7 に示す。

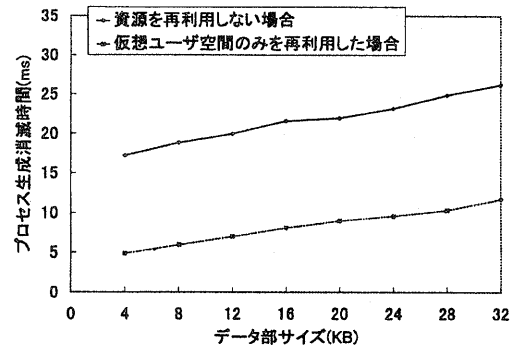


図 7 内容を利用しない仮想ユーザ空間の再利用効果 (テキスト部 4KB 固定)

図 7 より、プロセス生成消滅時間は、内容を利用しない仮想ユーザ空間を再利用することにより、資源を再利用しない場合に比べて、9ms 以上高速化できる。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できる。そのため、内容を利用しない仮想ユーザ空間を再利用する場合は、仮想領域と仮想空間も再利用していることになる。

仮想空間の再利用効果は一定であるので、内容を利用しない仮想ユーザ空間と仮想領域の再利用効果は、

図7の二つの場合の傾きの差分より、約0.29ms/4KBの時間短縮効果があることがわかる。

3.5.6 仮想領域の再利用効果

仮想領域を再利用した場合と資源を再利用しない場合について、データ部サイズとプロセス生成消滅時間の関係を、図8に示す。

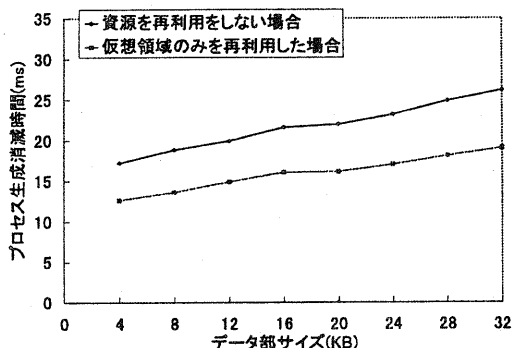


図8 仮想領域の再利用効果(テキスト部4KB固定)

図8より、プロセス生成消滅時間は、仮想領域を再利用することにより、資源を再利用しない場合に比べて、4.5ms以上高速化できる。表1に示したように、仮想領域の再利用効果は、その大きさに比例する。したがって、仮想領域を再利用した場合の再利用効果は、図8の二つの場合の傾きの差分より、約0.35ms/4KBの時間短縮効果があることがわかる。

内容を利用しない仮想ユーザ空間の再利用効果が、約0.29ms/4KBであるのに対し、仮想領域のみの再利用効果の方が、約0.35ms/4KBで効果が高い。先にも述べたように、内容を利用しない仮想ユーザ空間の再利用効果には、仮想領域の再利用効果も含まれるため、仮想領域のみの再利用効果よりも効果が大きいが予想される。しかし、内容を利用しない仮想ユーザ空間を再利用する場合のプロセスの生成処理では、テキスト部、データ部、BSS部、ユーザスタック部の条件に合う仮想ユーザ空間を一つ探す。次に、条件に合った仮想ユーザ空間が貼り付いている仮想空間に、貼り付いている他の仮想ユーザ空間について、再利用できるか否かを調べ、できなければ仮想空間から剥す。このような処理をしているため、仮想領域のみを再利用する場合に比べオーバーヘッドが大きく、再利用効果が小さくなったと考えられる。

3.5.7 プログラムの再利用効果

プログラムを再利用した場合と資源を再利用しない場合について、データ部サイズとプロセス生成消滅時間の関係を、図9に示す。図9より、再利用効果はわずかであることがわかる。

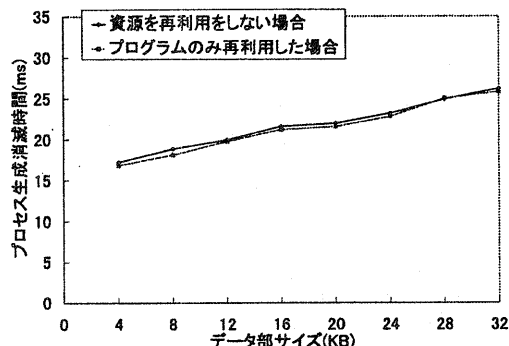


図9 プログラムの再利用効果(テキスト部4KB固定)

プロセスの生成処理で、プロセス管理は、プログラム管理に対しプログラムを確保するように要求する。次に、プログラム管理は、データ管理に対してプログラムのデータを外部記憶装置からメモリ中に読み込むように要求し、プログラムを解析する。プログラムを再利用することにより、データ管理に対し、プログラムのデータをメモリ中に読むように要求せず、プログラムを解析しないため、わずかに高速化できる。また、今回の測定では、データを外部記憶装置から読み込まない環境にしているため、図9の資源を再利用しない場合には、ディスクI/Oの時間は含まれていない。実際の環境では、プログラムのデータを外部記憶装置からメモリ中に読み込む際に、ディスクI/Oが生じる。プログラムを再利用すると、データをメモリ中に読み込む必要がなくなるため、ディスクI/Oの発生する環境でもディスクI/Oは発生しない。ディスクI/Oの時間は、メモリの操作にかかる時間に比べて、かなり大きい。したがって、プログラムの再利用の効果は大きいといえる。

3.5.8 仮想領域と仮想空間を再利用した場合と内容を利用しない仮想ユーザ空間の再利用効果

仮想領域と仮想空間を再利用した場合と仮想ユーザ空間を再利用した場合について、データ部サイズとプロセス生成消滅時間の関係を、図10に示す。

図10より、仮想領域と仮想空間を再利用した場合と仮想ユーザ空間を再利用した場合を比較するとほ

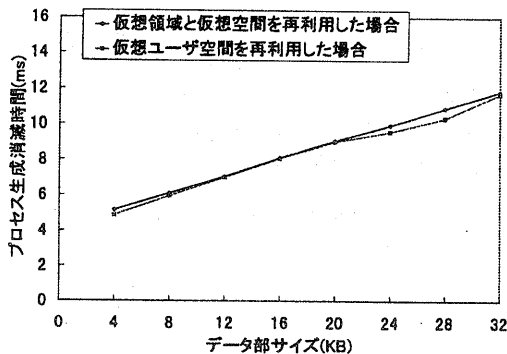


図 10 仮想領域と仮想空間を再利用した場合と内容を利用しない仮想ユーザ空間の再利用効果(テキスト部 4KB 固定)

とんど差がないことがわかる。この二つの場合の処理の違いは、仮想領域を仮想空間に貼り付けるかどうかという違いだけである。つまり、仮想領域を仮想空間に貼り付ける処理にはそれほど時間はかからないということがわかる。

ここで、資源を再利用できる頻度を考える。仮想領域は、大きさ情報のみで再利用できるのに対し、仮想ユーザ空間は、大きさとアドレス位置の情報が必要である。また、前者の場合は、仮想領域が再利用できなくても仮想空間が再利用できる場合もあり得るし、この逆もあり得る。したがって、仮想領域と仮想空間を再利用する場合は、わずかに効果が小さいものの、再利用できる頻度が大きいといえる。

同じプログラムファイルを使ってプロセスが何度も生成消滅する場合、仮想ユーザ空間を再利用する資源として指定すると、次にそのプロセスを生成する時に仮想ユーザ空間をすべて再利用できるため、効果が大きい。一方、あまり使われないプログラムファイルを利用したプロセスの仮想ユーザ空間を再利用するように指定すると、再利用される可能性が小さい。仮想領域と仮想空間を再利用する場合は、大きさ情報で再利用でき、再利用できる頻度は大きいので、あまり使われないプログラムのプロセスを消滅させる場合に、仮想領域と仮想空間を再利用するように指定すると資源が再利用されやすいと考えられる。

4 おわりに

プロセスの構成要素を資源として分離し独立化した *Tender* オペレーティングシステムのプロセス構造と、資源を再利用したプロセスの生成と消滅の高

速化について述べた。

Tender では、プロセスの各構成要素の関係を明確化し資源として分離し独立化させ、これにともない資源毎に管理表も分離した。

次に、プロセス生成の高速化は、事前に作り置きされた資源や再利用可能な資源を利用することにより実現でき、プロセスの消滅の高速化は、プロセスが利用していた資源を消滅させずに再利用するために保存しておくことにより実現できることを述べた。再利用可能な各資源について、再利用の可否の観点から明らかにし、各資源の再利用を個別に自由にできるように、各資源をビットに対応させたフラグで選択する再利用インタフェースを示した。実測と評価により、各資源を再利用した場合のプロセスの生成と消滅における高速化が可能であることを実証し、次のような効果があることを述べた。すべての資源を再利用することにより、プロセス生成消滅時間を 15ms 以上高速化でき、データ部サイズに対し約 0.35ms/4KB の時間短縮効果がある。ワーク領域用仮想カーネル空間の再利用により、1.45m 高速化できる。内容を利用するテキスト部用仮想ユーザ空間の再利用により、テキスト部サイズに対してプロセス生成消滅時間を一定にできる。内容を利用しない仮想ユーザ空間の再利用により、プロセス生成消滅時間を 9ms 以上高速化でき、データ部サイズに対し約 0.29ms/4KB 時間短縮効果がある。仮想空間の再利用により、6.91ms 高速化できる。仮想領域の再利用により、プロセス生成消滅時間を 4.5ms 以上高速化でき、約 0.35ms/4KB 時間短縮効果がある。プログラムの再利用により、わずかに高速化でき、ディスク I/O の発生をおさえることができる。

今後の課題としては、再利用する資源の効率的な運用法の検討がある。

参考文献

- [1] 谷口秀夫：“分散指向永続オペレーティングシステム *Tender*”，情報処理学会コンピュータシステムシンポジウム，シンポジウム論文集 Vol.95, No.7, pp.47-54(1995).
- [2] 谷口秀夫，田中徳穂：“*Tender* のプロセス管理構造”，情処研報，Vol.96, No.79, pp.109-114 (1996).
- [3] 長嶋直希，谷口秀夫，牛島和夫：“*Tender* におけるヘテロ仮想記憶の設計”，情処研報，Vol.97, No.56, pp.25-30 (1997).