データ部の初期化によるプロセス再起動機構の実現と評価

田端, 利宏
九州大学大学院システム情報科学研究科情報工学専攻 : 博士後期課程

谷口, 秀夫
九州大学大学院システム情報科学研究科情報工学専攻

https://doi.org/10.15017/1515706

出版情報：九州大学大学院システム情報科学紀要. 5 (2), pp.237-242, 2000-09-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第5巻 第2号 平成12年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.5, No.2, September 2000

データ部の初期化によるプロセス再起動機構の実現と評価

田端 利宏* ・ 谷口 秀夫**

Implementation and Evaluation of Mechanism of Restarting a Process by Initializing Data Segments

Toshihiro TABATA and Hideo TANIGUCHI

(Received June 16, 2000)

Abstract: Process creation and disappearance is a large load among processing of operating system. The research that speeds up process creation since before, is being done and sticky bit and vfork system call in the UNIX are materialized. For example, process creation and disappearance break out a repetition, because compiler is run many times in UNIX command of "make". Because of this, it is important when it speeds up the processing of operating system, to speed up process creation and disappearance. We perceive to the processing that delete a process and create the same process newly. And we propose mechanism of restarting a process by initializing data segments. The mechanism is able to run the process that ran it once from the beginning once again. This paper describes mechanism of restarting process by initializing data segments and reports performance of the mechanism, comparison with established operating system and evaluation by a real application.

Keywords: Create process, Operating system, Program, Recycle

1. はじめに

プロセスの生成と消滅の処理は, オペレーティングシステム(以降, OSと略す)の処理の中でも負荷の大きな処理である. プロセスの生成では, 仮想記憶空間を生成し, プログラムをメモリ空間に読み込む必要がある. このとき, メモリ間データコピーやディスクI/Oが発生するため, 負荷の大きな処理となっている. 例えば, プログラムのmake処理では, コンパイラが繰り返し実行されるため, プロセスの生成と消滅が繰り返し起こる. このため, プロセスの生成と消滅を高速化することは, 処理を高速化する上で重要である. 従来から, プロセスの生成を高速化する研究がなされており, UNIXではsticky bitやvforkシステムコールが実現されている[1]. また, プロセスを軽量化する研究も行なわれており, 軽量化プロセスやスレッドなどが実現されている[2)3)]. スレッドは, スレッド間で同じプログラムを共有するため, スレッド生成の負荷はプロセス生成よりも小さく, スレッド間で協調処理がしやすいという長所があるが, スレッド間の保護が弱いという短所がある.

我々は, プロセスを消滅させ新たに同じプロセスを生成するという処理に着目し, 一度走行させたプロセスを再び最初から走行できるようにするプロセスの再起動機構を提案する. プロセスの再起動機構を実現することで, 同じプロセスを繰り返す実行する場合にプロセスの生成と消滅の処理を高速化できる. プロセスの再起動機構を我々が開発しているTenderオペレーティングシステム[4]で実現し, 既存のOSとの評価を行なった.

本論文では, プロセスの基本構造について述べ, プロセスのデータ部の初期化による再起動機構の基本方式について述べる. Tender上で実現したプロセスの再起動機構の基本性能, 既存OSのプロセス生成との比較, および実アプリケーションによる評価結果を報告する.

平成12年6月16日受付

* 情報工学専攻博士後期課程
** 情報工学部門

2. プロセス管理

2.1 プロセスの構造

プロセスとは, OSがプログラムを実行するとき, 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, その状態を制御し, 終了させる. また, プロセスの状態を, プロセス管理表により管理する.

プロセスの構成要素を**Fig. 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(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

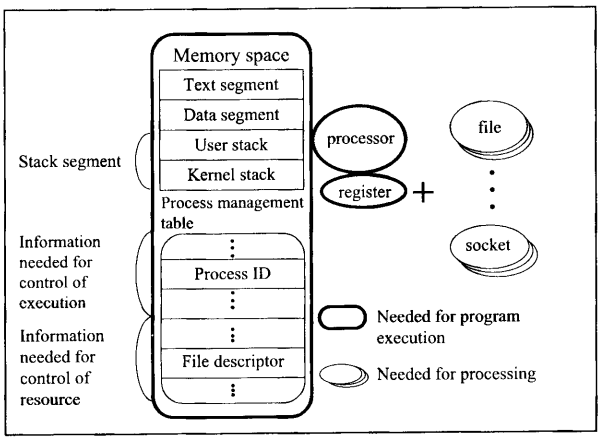

Fig.1　Elements of process.

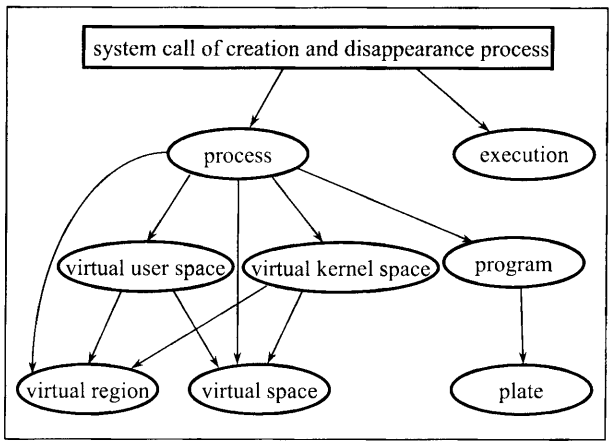

Fig.2　Relation of resources on process creation and disappearance.

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．

2.2　問題点

既存OSのプロセス管理には，大きな問題が二つある．一つは，プロセス管理情報を様々なモジュールが共有しているという問題である．様々なモジュールがプロセス管理情報を操作するため，動作の把握が困難であり，プロセス管理の情報が明確に分類されてないため，構造の理解が困難である．さらに，プロセス管理の機能の変更や追加が，さまざまなモジュールに影響を及ぼすため，その改版が困難なものになっている．もう一つの問題点は，プロセスの各構成要素の存在がプロセスの存在を必須としている点である．これは，プロセスの各構成要素の情報は，プロセス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，存在できないためである．このため，プロセスが利用していた資源をプロセス消滅時に解放してしまい，プロセス生成時に資源を再利用できないため，プロセスの生成と消滅時のオーバヘッドは大きくなっている．

*Tender*では，プロセスの構成要素を資源として分離し独立化させ，管理情報も資源毎に分離し，管理情報の共有をなくした．この結果，プロセスと構成要素の関係が明確になり，プロセスの構成要素は，プロセスの存在に関係なく存在することが可能となった．例えば，UNIXの場合，プロセスが消滅すると，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．*Tender*では，プロセス消滅時に指定したメモリ空間を残して，プロセス生成時に再利用することが可能である[5]．

3.　*Tender*オペレーティングシステム

3.1　資源の独立化機構

*Tender*ではOSの操作する対象を資源として，分離し独立化した．資源には，資源名と資源識別子を付与し，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を資源毎に分離し，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毎に分離し，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なうことで，資源の事前生成や保留により，資源の作成や削除を伴う処理を高速化している[5]．更に，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の資源の独立化の例として，プロセスの生成と消滅時の資源の処理の呼びだし関係を**Fig. 2**に示す．**Fig. 2**中の矢印は，資源間の処理の呼びだし関係を表している．

3.2　資源「プログラム」

資源「プログラム」は，プログラムの「テキスト部の大きさ」，「テキスト部の先頭番地」，「データ部の大きさ」，「データ部の先頭番地」，および「プログラムの開始番地」の情報から成り，プログラムデータの実行形式を隠蔽しており，プログラムをメモリ中に存在させる．

この結果，プロセス管理は，プログラムデータの内容を意識することなく処理を実行することができる．また，プログラムを資源化したことで，プロセスの存在に関係なくプログラムが存在できるため，既にメモリ中に存在するプログラムを利用してプロセスの生成を高速化できる．

3.3　資源「演算」

*Tender*では，プロセッサの割当単位を資源として分離・独立化し，「演算」と名付けた[6]．資源「演算」とは，プロセッサを利用でき，プログラムを実行できる程度(演算の

Table-1 Interface of restart process.

type	arguments
procreset(pid, argv)	pid: specify process for restart. argv: pointer for arguments.

程度と名付ける)を持つ資源である. 演算をプロセスと関連付けることにより, プロセスの実行が可能になる. つまり, 利用者は, プロセスと演算を生成し, プロセスに演算を関連付けることで, 演算が持つ演算の程度に応じてプロセスを走行させることができる. このため, 演算の程度を変更することで, プロセッサを利用できる程度を変更できる. これにより, プロセスを生成し, 演算の関連付けを行なわないことによるプロセスの作り置きや, 演算の関連付けと関連付けの解除によるプロセスの実行の停止と再開が容易に実現できる.

4. プロセスの再起動機構

プロセスの再起動とは, 任意の時点でプロセスの実行を停止し, プロセスのデータ部を初期化し, プロセスの管理表を初期化することで, プロセスを再び最初から実行させることである.

4.1 基本機構

プロセスの再起動では, 二つの処理を行なう. 一つは, プロセスの実行を停止することである. *Tender*では, 任意の位置でのプロセスの実行の停止は, 資源「演算」を実現したことにより, 容易に実現できる. 二つめは, データ部を初期化しプロセスを最初から実行できる環境を構築するための処理を行う. プロセスの再起動機構をシステムコールで実装した. プロセスの再起動のインタフェースをTable-1に示し, 処理内容を説明する.

(1) OSはシステムコールの引数pidとargvを受け取る.
(2) プロセスpidの各部分(テキスト部, データ部, BSS部)のアドレス位置と大きさを取得する.
(3) 仮想記憶空間を再起動させるプロセスのものに切替え, 取得したプロセスの情報をもとにして, データ部とBSS部の内容を初期化し, プロセスに渡す引数argvをユーザスタックに複写する.
(4) 仮想記憶空間を元のものに切替え, プロセスpidのプロセス管理表のエントリを初期化する.

4.2 特徴と利点

プロセスの再起動を行うことにより, 以下の特徴と利点がある.

(1) 同じプログラムを繰り返し実行する場合に, プロセスを作り直さずに再起動することで, 高速に処理を開始することが可能である.
(2) プロセスを作り直すと, 利用していた資源を解放し, 生成し直さなければならない. プロセスの再起動では, 再起動前に利用していた資源を, 再起動後にそのまま利用でき, 資源を確保するオーバヘッドを減らすことができる.
(3) プロセスの再起動の前後では, プロセス識別子は不変である. このため, 他のプロセスとの処理をそのまま継続できる. 例えば, 通信では再接続なしにそのまま相手と通信できる.

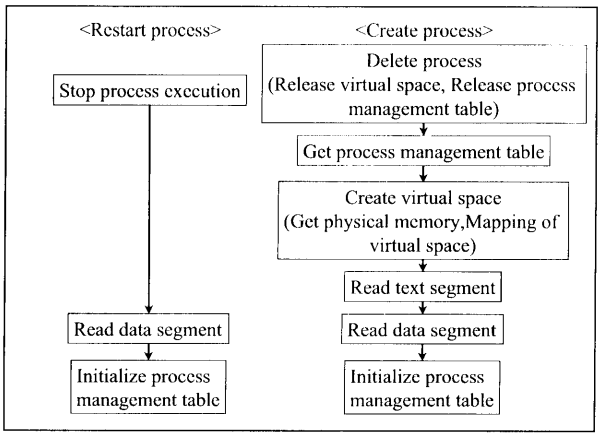

Fig.3 Processing of restart process and create process.

4.3 プロセスの再起動と生成の処理比較

プロセスの再起動機構とプロセスの生成処理の比較を行う. プロセス再起動処理とプロセス生成処理を**Fig. 3**に示す. プロセスの再起動は, プロセスの実行を停止し, データ部の読み込みを行ない, プロセス管理表の初期化を行ないプロセスを最初から走行させる. プロセスの生成処理は, プロセスを削除の後, 新たに確保するプロセスのエントリをプロセス管理表に確保する. それから, 空間の作成を行ない, テキスト部とデータ部を読み込み, プロセス管理表を初期化することで新しいプロセスを生成する.

プロセス生成処理は, プロセス生成時にメモリ空間にプログラムを読み込む方式(メモリ常駐)とプロセス生成時にはメモリ空間にプログラムを読み込まず, プロセスの実行時にページ例外により必要ページのみ読み込む方式(On Demand Paging(ODP))の二つの方法がある.

プロセスの再起動は, メモリ常駐のプロセスの生成方法に比べ, プロセスの削除, プロセス管理表のエントリの確保, 空間の作成, およびテキスト部の読み込みを行なわないので高速である. また, プロセスの再起動は, 資源「プログラム」を利用しているので, プログラムを空間に読み込む際には, ディスクI/Oは発生しない. ODPを採用したプロセス生成法では, プロセスの生成時に実メモリの確

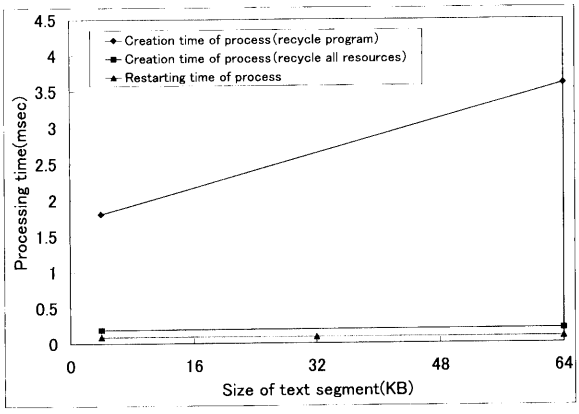

Fig.4 Relation of text segment size and processing time.

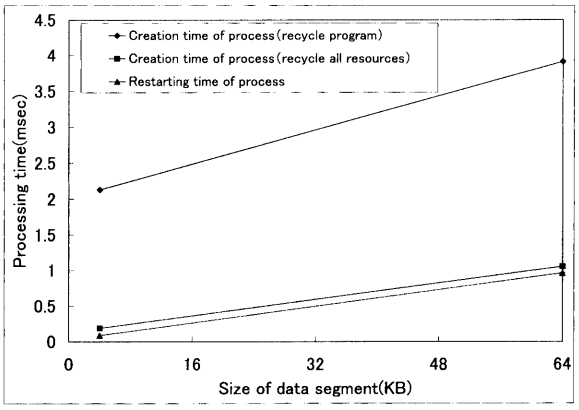

Fig.5 Relation of data segment size and processing time.

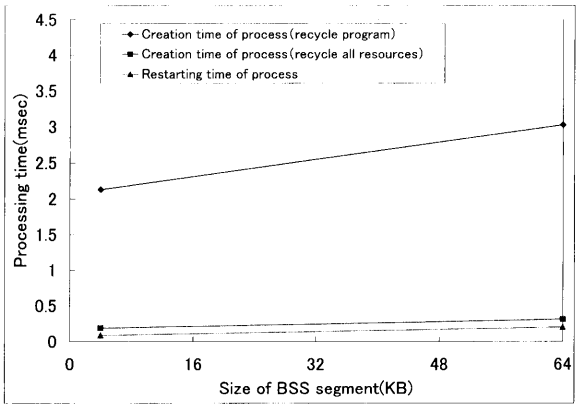

Fig.6 Relation of BSS segment size and processing time.

保, テキスト部の読み込み, およびデータ部の読み込みを行なわない分, メモリ常駐の方式より高速である. しかし, プロセス実行時にページフォールトが発生し, プログラムの内容を読み込むため, プロセスの実行時の処理は遅い. ただし, 必要最小限のメモリしか必要としないという長所がある.

5. 評　　価

5.1 基本性能

プロセスの再起動機構の基本性能の評価を行なうために, *Tender*上でプロセスの再起動時間とプロセスの生成と消滅時間を測定した. *Tender*では, プロセスを生成する際に, プロセスを構成する資源を再利用できる[5]ので, すべての資源を再利用した場合と資源を再利用しない場合の二つの場合の測定を行なった. ディスクI/Oによる影響をなくすために, 常に資源「プログラム」を再利用した.

プロセスAがプロセスBを再起動する時間と生成消滅する時間を測定した. プロセスの再起動の処理時間の測定は, プロセスBが実行を停止した状態で, プロセス再起動のシステムコールをプロセスAが発行し, 再起動処理が終了するまでの時間を測定した. プロセスの生成と消滅も同様に, プロセスAが実行を停止したプロセスBを消滅させ, 同じプロセスを生成する処理が終了するまでの時間を測定した.

測定に使用した計算機は, PentiumII 450MHzの計算機で, ハードウェアクロックのカウンタを利用して処理時間を測定した.

テキスト部の大きさを可変にし, データ部とBSS部を4KBに固定した場合の測定結果を**Fig. 4**に示す. プロセス再起動処理とすべての資源を再利用したプロセスの生成消滅処理は, テキスト部をそのまま利用できるため, テキスト部の大きさにかかわらずほぼ一定の処理時間(約90μsec, 約200μsec)である. プロセスの再起動処理は, プロセスの実体を消滅させないため, すべての資源を再利用したプロセスの生成消滅処理に比べ, 約110μsec高速である. プログラムを再利用したプロセスの生成消滅処理は, プロセス生成時にテキスト部の内容を読み込むためのメモリ間コピーが必要であるため, テキスト部の大きさに比例した処理時間がかかる.

データ部の大きさを可変にし, テキスト部とBSS部を4KBに固定した場合の測定結果を**Fig. 5**に示す. すべての場合で, データ部の内容を読み込むためのメモリ間コピーが必要であるため, データ部の大きさに比例した処理時間がかかる. ただし, プロセス再起動処理とすべての資源を再利用したプロセスの生成消滅処理は, データ用の領域の空間へのマッピング処理が必要ないため, プログラムを再利用したプロセスの生成消滅よりも, グラフの傾きが緩やかである. プロセス再起動処理は, プロセスの実体を消滅させないため, すべての資源を再利用したプロセスの生成消滅処理に比べ, 約100μsec高速である.

BSS部の大きさを可変にし, テキスト部とデータ部を4KBに固定した場合の測定結果を**Fig. 6**に示す. BSS部は, 初期値を持たない変数の集合部であるため, 値0で内容を初期化する. このため, メモリ間コピーの必要なデー

タ部の初期化よりも処理が高速であるが, すべての場合でBSSの大きさに比例した処理時間がかかる. プロセス再起動処理とすべての資源を再利用したプロセスの生成消滅処理は, BSS用の領域の空間へのマッピング処理が必要ないため, プログラムを再利用したプロセスの生成消滅よりも, グラフの傾きが緩やかである. プロセスの再起動処理は, プロセスの実体を消滅させないため, すべての資源を再利用したプロセスの生成消滅処理に比べ, 約100μsec高速である.

プロセス再起動時間(y_1), すべての資源を再利用したプロセスの生成消滅時間(y_2), およびプログラムを再利用したプロセスの生成消滅時間(y_3)を測定結果から定式化した結果を以下に示す. テキスト部, データ部, およびBSS部の大きさ(KB)をx_1, x_2, x_3とする.

$$y_1(\mu sec) = 14.5x_2 + 2x_3 + 35 \quad (1)$$

$$y_2(\mu sec) = 14.5x_2 + 2x_3 + 130 \quad (2)$$

$$y_3(\mu sec) = 30x_1 + 30x_2 + 15x_3 + 1850 \quad (3)$$

5.2 既存OSとの比較

プロセスに関する処理の流れは, プロセス生成処理とプロセスを実行し終了するまでの処理の二つに分けて考えることができる. まず, プロセス生成処理について考える.

UNIXでは, fork/vforkシステムコールにより, プロセス管理表の初期化, 仮想記憶空間とメモリ空間の作成を行ない, execシステムコールでプロセス実行環境を生成する. fork/vforkシステムコールは, copy on write技術を用いているので, 実際にメモリ空間を作成せず, 親プロセスと子プロセスが同じ空間を共有する. 実際に, メモリ空間に書き込みが起こった場合に, 実メモリを確保し内容をコピーする. execシステムコールでは, ODP技術を用いているので, メモリ空間にプログラムを読み込む処理を行なわず, プロセス実行時にメモリにアクセスしページ例外が起こった場合に, メモリにプログラムの内容を読み込む. このため, プロセスサイズに関係なくほとんど一定時間でプロセス生成処理が終了する. 5.1節と同じ計算機で, UNIXの一つであるBSD/OS上でvfork/execシステムコールの処理時間を測定した. 親プロセスが, vforkシステムコールを発行し, waitシステムコールにより子プロセスの実行終了を待ち制御が戻ってくるまでの時間を測定した. 子プロセスは, execシステムコールを発行し, 終了する. このときの処理時間は, 約1700μsecであった. したがって, プロセス再起動時間を式(1)で算出した値が, BSD/OSのプロセス生成時間1700μsecを越えないプログラムについては, プロセス再起動機構が高速であるといえる.

次にプロセスの実行し終了するまでの処理について考える. *Tender*では, プロセス生成時に, プログラムの内容をすべてメモリ空間に読み込むため, ページ例外なしにプロセスを実行できる. ただし, ページがスワップアウトされた場合には, ページ例外が発生する. これに対し, BSD/OSでは, ODP技術を用いているため, プロセスの実行時にメモリアクセスによりページ例外が発生する. ページ例外処理では, ディスクから内容を読み込むこともある. ディスクI/Oは数msecの処理時間がかかるため, プロセス実行の際の大きなオーバヘッドとなることが予想される. このため, *Tender*のプロセスの実行の方が高速である. しかし, *Tender*では, メモリ中にすべてのプロセスを常駐させるため, メモリを多く消費するという短所がある. 例えば, BSD/OSカーネルのmake処理で繰り返し利用されるプログラム(cpp, as, cc1, gcc2)をメモリに常駐させたときのメモリサイズは約1250KBとなる.

5.3 実アプリケーションによる評価

プロセスの再起動機構の効果が実際にどの程度あるのかを評価するために, 実アプリケーションのプロセス生成のログを取り, プロセスの再起動機構の評価を行う. BSD/OS version 3.1のカーネルを作成するmake処理についてプロセス生成のログを取った. カーネルのmake処理では, ソースファイルをコンパイルするために, コンパイラが繰り返し呼ばれ, プロセスの生成と消滅を繰り返している. 生成されるプロセスのプログラム名とその大きさ(byte)と生成回数をTable-2に示す.

Table-2と同じ大きさのプログラムを同じ回数だけ生成する処理を, *Tender*上でプロセスの再起動で行なう場合, すべての資源を再利用してプロセスの生成を行なう場合, およびプログラムのみを再利用してプロセスの生成を行なう場合について処理時間を5.1節の近似式から算出した. 結果をTable-3に示す. Table-3より, プログラムのみを再利用した場合の処理時間(12.0 秒)に比べ, プロセスの再起動した場合の処理時間(0.36秒)は, 約33.3倍高速であることがわかる. プログラムを再利用した場合は, プログラムが必ずメモリ上に存在するため, ディスクI/Oが起こらないが, プログラムを再利用しない場合では, ディスクI/Oが発生する場合があるため, その場合と比較するとプロセスの再起動の効果はさらに大きくなる. Table-3より, すべての資源を再利用した場合の処理時間(0.47秒) に比べ, プロセスの再起動の処理時間(0.36秒)は約1.3倍高速であることがわかる. これは, プロセスの実体を消さずにそのまま利用してプロセスを再起動させるためである.

次に, BSD/OSでのプロセス生成処理と比較する. BSD/OSでのカーネルのmakeでのプロセス生成回数は1155回である. 5.2節で測定した一回当りのBSD/OSのプロセス生成処理時間から, プロセスの生成にかかる時間を算出すると1700$\mu sec \times 1155 = 1.96$秒となる. これから,

Table-3 Processing time in making BSD/OS kernel.

	restart process	create process(recycle all resources)	create process(recycle program)
processing time(sec)	0.36	0.47	12.0

Table-2 Programs executed in making BSD/OS kernel.

name	text	data	BSS	times
cpp	36864	4096	13092	282
as	53248	16384	34964	277
cc1	970752	40960	67324	272
gcc2	40960	4096	0	272
sh	118784	8192	3348	14
[	3472	184	72	11
rm	2800	140	28	8
ln	912	140	16	4
ar	12288	4096	0	2
ld	28672	4096	620	2
cat	1956	132	36	1
chmod	1348	140	0	1
date	3204	336	12	1
expr	3876	2256	24	1
hostnname	488	140	0	1
mv	1500	136	8	1
pwd	360	140	0	1
ranlib	5136	288	380	1
size	3516	180	360	1
strip	2480	136	0	1
touch	2588	128	0	1
total				1155

プロセスの再起動機構の方が, 約5.4倍高速であることがわかる.

6. ま　と　め

本論文では, プロセスのデータ部を初期化することによるプロセスの再起動機構を提案した. プロセスの再起動機構は, 高速に処理を開始でき, 再起動前に利用していた資源を継続して利用でき, プロセスIDが不変なのでそのまま処理を継続可能であるという特徴がある. プロセスの再起動機構と既存OSのプロセス生成処理を比較し, 以下のことを明らかにした. プロセスの再起動は, プロセスの削除, プロセス管理表のエントリの確保, 空間作成, およびテキスト部の読み込みの各処理を行なわないので高速である. また, プログラムをメモリ上に常駐できる資源「プログラム」を利用しているため, プロセス再起動時とプロセス実行時にディスクI/Oは発生しない. ただし, その分だけ実メモリを多く利用する.

基本性能を測定し, プロセスの再起動時間(μsec)は, $14.5\times$(データ部サイズ(KB))$+2\times$(BSS部サイズ(KB))$+35$となることを明らかにした.

また, UNIXの一つであるBSD/OSのプロセス生成処理と比較し, $1700(\mu sec) \geq 14.5$(データ部サイズ(KB))$+2$(BSS部サイズ(KB))$+35$が成り立つプログラムについては, プロセスの再起動機構が高速であることを明らかにした. また, プロセスの再起動では, プロセス実行時にページ例外が発生しないため, ODPを採用しているBSD/OSよりも高速にプロセスを実行できる. 実アプリケーションの例として, BSD/OSカーネルを作成するmake処理のプロセス生成処理についてプロセス生成時間を算出し, *Tender*上に実装したプロセス再起動機構と比較した. この結果, プロセスの再起動機構は, *Tender*上での通常のプロセス生成に比べ, 約33.3倍高速で, BSD/OSに対しては約5.4倍高速であることを明らかにした.

今後の課題としては, プロセスの再起動時にプロセスの確保していた資源の処理に関する検討がある.

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) 横山 和俊, 箱守 聡, 谷口 秀夫 : "処理形態に適したプロセスの軽量化," 情処研報, Vol.91, No.63, p.33-40 (1991).
3) Avadis Tevanian, Jr. Richard, F. Rashid, David B. Golub, David L. Black, Eric Cooper, and Michael W. Young : "Mach Threads and the Unix Kernel: The Battle for Control," Technical Report CMU-CS-87-149, School of Computer Science, Carnegie Mellon University, August (1987).
4) 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).
5) 田端 利宏, 谷口 秀夫 : "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," 情処研報, Vol.98, No.33, pp.79-86 (1998).
6) 田端利宏, 谷口秀夫 : "*Tender*オペレーティングシステムの資源「演算」によるプログラム実行速度調整機能の実現と評価", 情報処理学会論文誌, Vol.40, No.6, pp.2523-2533 (1999).