オペレーティング・システム　56－8
(1992. 8. 19)

多数プロセス生成の高速化手法

横山　和俊　　谷口　秀夫
ＮＴＴデータ通信（株）開発本部

並列処理を行なう環境においては、オペレーティング・システム（ＯＳ）が並列処理の単位であるプロセスを効率良く制御することが必要である。ＯＳのプロセス制御機能のうち、プロセス生成処理は、ディスクＩ／Ｏ処理を伴うため、処理時間が大きい。特に、並列処理中にプロセス数や実行内容（プログラム）が動的に変化する場合、プロセスの生成処理が頻繁に発生する。そのため、多数のプロセスを生成する処理を高速に行なうことは、システムの性能向上のため重要である。本稿では、並列処理の動的な変化を効率良く制御するため手段として、多数プロセスの生成を高速化する。

Fast Mechanism on Multi-process Creation

Kazutoshi YOKOYAMA , Hideo TANIGUCHI
Development Headquarters
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION
Kowa Kawasaki Nishi Bldg. 66-2 Horikawa-cho, Saiwai-ku,
Kawasaki-shi, Kanagawa 210, Japan
JUNET: yokoyama@rd.nttdata.jp

Processes are considered as units of the concurrent processing. It is important to control processes efficiently. The process creation is usually slow due to the disk I/O processing. In the case that the process number and the programs change dynamically during the concurrent processing, the process creation happens frequently. That is, the fast facility of the process creation is required. In this paper, a fast mechanism of the multi-process creation to control the concurrent processing with the dynamic change is proposed.

1．はじめに

マイクロプロセッサの高性能化により、負荷の大きい処理を高速に行なうことが可能になってきている。要求される処理を効率良く行なうためには、同時に実行可能な処理を並列に行なうことが要求される。並列処理の一つの分野である多重プログラミングにおいては、並列処理単位であるプロセスを効率良く制御するプロセス制御方式の確立が重要である。

多重プログラミング環境を提供するオペレーティング・システム（以降、ＯＳと略す）において、処理を並列に実行する場合、ＯＳのプロセス生成終了処理やプロセス切換処理およびプロセス間の同期処理が頻繁に実行される。これらの処理のうち、プロセスの生成処理は、ディスクＩ／Ｏ処理を伴うため、処理時間が大きい。一方、文献[1]で述べられているように、並列処理中に必要な実行単位（プロセス）の数や実行内容（プログラム）が動的に変化する場合が考えられる。このような場合には、プロセスの生成処理が頻繁に発生することになる。そのため、多数プロセスを生成する処理を高速に行なうことは、システムの性能向上のため重要である。

プロセスの生成終了処理や切換処理などを効率良く行なう１つの手法として、プロセスを軽量化する方法がある。（以降では、軽量化されたプロセスを軽いプロセスと呼ぶ。）この方法は、プロセス間でデータや資源を共有することにより、１つのプロセスが保有する情報を少なくし、プロセスの生成や切換の操作を高速に行なうことを試みている。また、別の試みとして、ＤＩＲＯＳ（DIstributed Real-time OS）[2]では、マルチプロセス生成機能を提案している[3]。この機能では、プロセスの軽量化を行なわず（以降、従来プロセスと呼ぶ）、多数プロセス生成の高速化を図っている。

軽いプロセスとマルチプロセス生成機能の試みでは、高速な生成処理の可能な処理分野が限られており、プロセス数や実行プログラムが動的に変化する処理に対応することは困難である。本稿では、多重プログラミング環境で並列処理を行なう際に重要な機能である多数プロセス生成処理について、高速な手法を提案する。

2．プロセス生成処理

2.1　並列処理モデル

本稿で対象とする並列処理モデルを、図１に示す。図１に示すように、処理の開始から終了までの間に、処理に必要なプロセス数や実行プログラム（以降、これらをまとめて並列度と呼ぶ）が動的に変化する。並列度の動的な変化には、以下に示す２つの生成形態が考えられる。

（形態Ａ）プロセス数の変化が実行プログラムの変化を伴う場合

（形態Ｂ）プロセス数の変化が実行プログラムの変化を伴わない場合

図中の（ａ）で示した部分は、（形態Ａ）の場合である。１個のプロセスがプログラムＡを実行する状態からｍ個のプロセスがプログラムＢを実行する状態に変化する。図中の（ｂ）で示した部分は、（形態Ｂ）の場合である。ｍ個のプロセスがプログラムＢを実行していた状態からｎ個のプロセスがプログラムＢを実行する状態に変化する。

2.2　多数プロセスの生成方式

１つのプログラムから多数のプロセスを生成する従

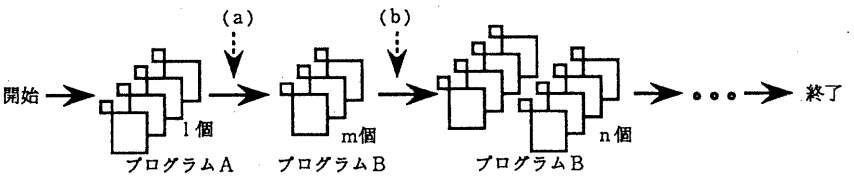

図１　並列処理モデル

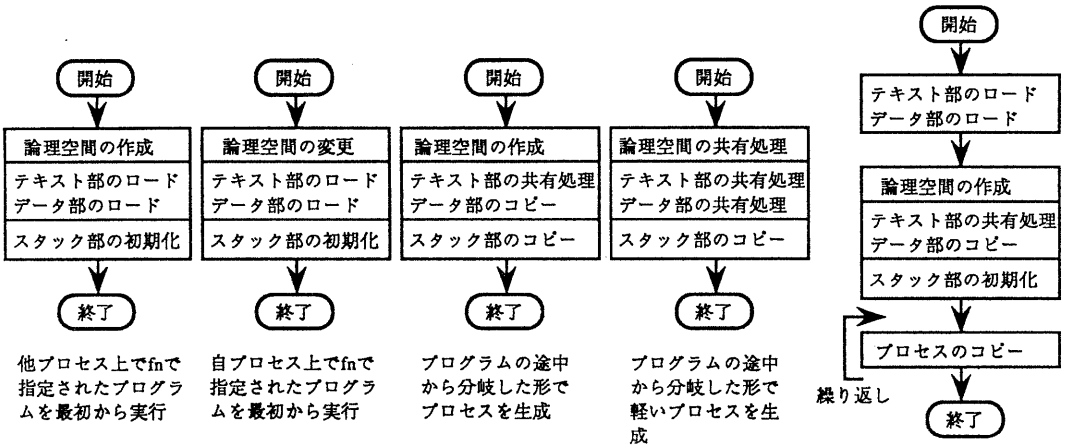

図2　生成処理方式

来の方式について、以下に説明する。（生成するプロセス数をｎとする。）

各方式の基本的なプロセス生成処理を図２に示す。多数プロセスの生成は、新規のプログラムを利用してプロセスを生成する形式（形式αと呼ぶ）と走行中のプロセスと同じプログラム利用してプロセスを生成する形式（形式βと呼ぶ）の２つに分類できる。

（1）従来プロセスの生成

ＤＩＲＯＳにおけるプロセス生成では、（形式α）と（形式β）の両形式に対して、

①ｎ回のcreprの繰り返し方式

が提供される。

ＵＮＩＸのプロセス生成では、（形式α）に対して、

②exec＆ｎ回のforkの繰り返し方式、

（形式β）に対して、

③ｎ回のforkの繰り返し方式

が提供される。

（2）軽いプロセス

軽いプロセスの代表的なものとして、Ｍａｃｈ[4]などのＯＳで採用されている、論理空間とテキスト部とデータ部と資源アクセス管理テーブルを共有したものが挙げられる。このプロセス生成では、（形式α）に対して、

④exec＆ｎ回のfork_LWPの繰り返し方式、

（形式β）に対して

⑤ｎ回のfork_LWPの繰り返し方式

が提供される。

（3）マルチプロセス生成機能

この生成方式では、プロセスに共通な情報の有効利用を図り、高速化を実現している。具体的には、ｎ個のプロセスのうち最初に１個のプロセスを生成する。残りのプロセスはそのプロセスをコピーすることで生成する。そのため、２個目以降のプロセス生成では、ディスクＩ／Ｏ処理は発生しない。この生成方式では（形式α）と（形式β）に対して、

⑥crempr方式

が提供される。

２．３　並列処理モデルへの適用

前節で示した①から⑥までの６種類の多数プロセスの生成方式について、各方式を並列処理モデルに適用したときの比較を表１に示す。表１より以下のことがわかる。

（1）（形態Ａ）に最も高速に対応できる生成方式は、⑥のcrempr方式である。次に、方式②、方式④、方式①の順である。方式④が低速であるのは、新規プログラムの起動のときのディスクＩ／Ｏ処理時間が大きいためである。これは、軽いプロセスのロード

表1　従来の生成方式の並列処理モデルへの適用性

方式	(形態A)			(形態B)		
	処理内容		速度	処理内容		速度
	ﾃﾞｨｽｸI/O処理	ﾒﾓﾘｺﾋﾟｰ処理		ﾃﾞｨｽｸI/O処理	ﾒﾓﾘｺﾋﾟｰ処理	
①n回のcreprの繰り返し	ﾃｷｽﾄ部：1回 ﾃﾞｰﾀ部：n回	—	最も低速	ﾃﾞｰﾀ部：n回		最も低速
②exec＆n回のforkの繰り返し	ﾃｷｽﾄ部：1回 ﾃﾞｰﾀ部：1回	ﾃﾞｰﾀ部：n回 ｽﾀｯｸ部：n回	やや高速	ﾃﾞｰﾀ部：1回	ﾃﾞｰﾀ部：n回 ｽﾀｯｸ部：n回	やや低速
③n回のforkの繰り返し	対処不可能		—	—	ﾃﾞｰﾀ部：n回 ｽﾀｯｸ部：n回	高速
④exec＆n回のfork_LWPの繰り返し	ﾃｷｽﾄ部：1回 ﾃﾞｰﾀ部：1回 (ｻｲｽﾞがほぼn倍)	ｽﾀｯｸ部：n回	低速	ﾃﾞｰﾀ部：1回 (ｻｲｽﾞがほぼn倍)	ｽﾀｯｸ部：n回	低速
⑤n回のfork_LWPの繰り返し	対処不可能		—	—	ｽﾀｯｸ部：n回	最も高速
⑥crempr	ﾃｷｽﾄ部：1回 ﾃﾞｰﾀ部：1回 (ｻｲｽﾞがほぼn倍)	ﾃﾞｰﾀ部：n回	最も高速	ﾃﾞｰﾀ部：1回	ﾃﾞｰﾀ部：n回	やや高速

生成プロセス数：n

モジュールサイズが、データ部を共有しているため増大することに起因する［問題1］[5]。また、最も高速である方式⑥においても、プログラムロードのためのディスクI／O処理が必ず発生する［問題2］。

（2）（形態B）に最も高速に対応できる生成方式は、⑤のfork_LWPの繰り返し方式である。次いで、方式③、方式⑥、方式②、方式④、方式①の順である。これは、方式⑤と方式③はすべてメモリ処理であるのに対し、方式⑥と方式②と方式④と方式①はディスクI／O処理が発生するからである［問題3］。

3．高速化の方針と課題

3.1　プロセス形態の決定

プロセス形態として、以下の理由により、従来プロセスを採用する。

（1）軽いプロセスのロードモジュールサイズを小さくすることは困難である。したがって、［問題1］を解決できない。

（2）軽いプロセスの形態では、プロセス間の不当アクセス防止が困難である。

また、対象とする並列処理モデルでは、動的に並列度が変化する。つまり、実行プログラムの変化を伴う（形態A）と伴わない（形態B）の両形態が存在する。そのため、（形態A）と（形態B）の両方での高速化が求められる。ここで、方式③は（形態A）に対応できない。したがって、以降では表1に示す、方式①、方式②、方式⑥、および（形態A）における方式②と（形態B）における方式③を組み合わせた方式、を対象にした高速化を述べる。

3.2　課題

3.2.1　生成処理の形式

多数プロセス生成処理の形式は、応用プログラムに対して、統一的な形式で提供することが望ましい。このとき、生成処理の性能を考慮することが必要である。

3.2.2　実現方式

残された2つの問題に基づく課題を以下に述べる。

（1）［問題2］への対応

以下の3つの課題がある。

(A) ディスクＩ／Ｏ時間の向上

複数のディスクを用いた並列読み込み方式やプログラムのディスク格納形式を検討し、ディスクからメモリへ転送する時間を短縮し、高速化を図る。

(B) ＣＰＵ処理とディスクＩ／Ｏ処理の並列化

ＣＰＵ処理とディスクＩ／Ｏ処理が並列動作可能な部分を抽出し、並列動作を行なうことにより、高速化を図る。

(C) ディスクＩ／Ｏ回数の削減

プログラムをディスクからメモリに一括転送する方式やプロセス生成に必要な情報（テキスト部やデータ部のディスク格納情報など）を有効利用する方式を検討し、ディスクＩ／Ｏ処理の回数を削減することで高速化を図る。

（2）［問題3］への対応

上記の(1)－(C)と同様な課題がある。

4．対処方式

3.2節で挙げた課題に対する対処方式について述べる。

4.1 生成処理の形式

生成処理の形式として以下の4つを考える。（生成するプロセス数をnとする。）

(A) ①n回のcreprの繰り返し方式

(B) ②exec & n回のforkの繰り返し方式

(C) ⑥crempr方式

(D) ⑦（形態A）における方式②と（形態B）における方式③の組み合わせ

各形式の比較を表2に示し、以下に説明する。

＜インタフェース＞

①と②と⑥は、（形態A）と（形態B）に対して同じインタフェースを応用プログラムに提供できる。

⑦の場合、（形態A）と（形態B）ではインタフェースが異なる。

＜性能＞

①と②と⑦では、creprとforkを繰り返し用いるため、生成処理中にユーザレベルで走行する処理が介在する。一方、⑥ではカーネルレベルで生成処理が走行する。そのため、⑥が高速である。

これらの理由により、生成処理方式として、crempr方式を採用する。したがって、現行のcrempr方式の高速化を実現する。

4.2 実現方式

3.2.2項で述べた課題のうち、(1)－(A)については、プログラムファイル連続領域への格納などの方式がある。(1)－(B)に関しては、非完了処理を用いた実現方式がある。そのため、ここでは、(1)－(C)と(2)のディスクＩ／Ｏ回数の削減について具体的な実現方式を述べる。

4.2.1 マルチプロセス生成処理の分析

現行のマルチプロセス生成処理の概要を図3に示し、（1）から（5）の処理を以下に説明する。

（1）ディスクＩ／Ｏ処理

表2　生成処理形式の比較

			(A) 方式①	(B) 方式②	(C) 方式⑥	(d) 方式⑦
ｲﾝﾀﾌｪｰｽの統一性	形式	(形態A)	n回のcreprの繰り返し	exec & n回のforkの繰り返し	crempr	exec & n回のforkの繰り返し
		(形態B)	n回のcreprの繰り返し	exec & n回のforkの繰り返し	crempr	n回のforkの繰り返し
	統一性		○	○	○	×
生成処理の性能	処理内容	(形態A)	ﾕｰｻﾞﾚﾍﾞﾙが存在	ﾕｰｻﾞﾚﾍﾞﾙが存在	ｶｰﾈﾙﾚﾍﾞﾙ	ﾕｰｻﾞﾚﾍﾞﾙが存在
		(形態B)	ﾕｰｻﾞﾚﾍﾞﾙが存在	ﾕｰｻﾞﾚﾍﾞﾙが存在	ｶｰﾈﾙﾚﾍﾞﾙ	ﾕｰｻﾞﾚﾍﾞﾙが存在
	速度		×	×	○	×

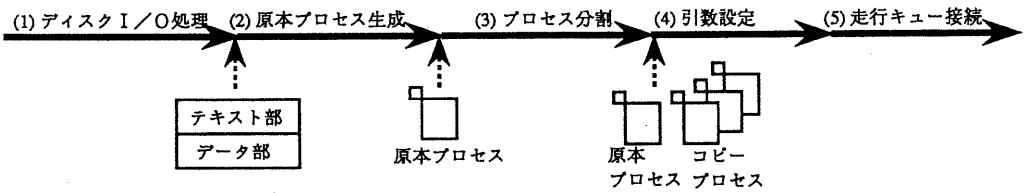

図３　マルチプロセス生成処理の流れ

プログラムをディスクからメモリ上にロードする処理である。

（２）原本プロセス生成

最初のプロセスを生成する。このプロセスは、残りのプロセスを生成するときのコピー元として使用される。（このプロセスを原本プロセスと呼ぶ）

（３）プロセス分割

上記の（２）で生成された最初のプロセスからメモリコピー処理で残りのプロセスを生成する。（このプロセスをコピープロセスと呼ぶ。）

（４）引数設定

各プロセスに引数を設定する。

（５）走行キューに接続

走行キューにプロセス群を接続する。

４．２．２　高速化の手段

多数プロセスの生成を高速化する手段として、「プロセス生成に必要な情報をメモリに常駐させる方式」を提案する。この方式においては、以下の２つの課題がある。

（１）常駐対象

プロセス生成に必要な情報のうち、メモリ上に常駐させる対象を決定する必要がある。対象とする常駐範囲は以下の２つである。

(A) テキスト部とデータ部

(B) テキスト部、データ部、スタック部、および管理テーブル

(A)の方が常駐に必要なメモリ量は少ない。しかし、生成処理の処理量が多くなる。

（２）利用方式

常駐した情報を利用する方式は、プロセス生成処理の開始点と関係が深い。プロセス生成処理の開始点として、以下のものがある。

❶システムコール発行の直後

❷ディスクＩ／Ｏ処理の直後

❸原本プロセスの生成の直後

❹プロセス分割の直後

開始点が後になるほど、高速なプロセス生成処理が可能である。

４．２．３　具体的な高速化方式

メモリに常駐させるプロセス生成情報とプロセス生成処理の開始点との組み合わせを考慮すると、５種類の高速化方式が考えられる。現行方式と高速化できる各方式の概要を図４に示し、以下に説明する。

（０）現行方式（常駐対象：なし、開始点：❶）

常に、ディスクＩ／Ｏ処理からプロセス生成を開始する。

（１）方式１（常駐対象：(A)、開始点：❷）

１回目のプログラム起動時にテキスト部とデータ部の常駐処理を行なう。２回目以降では、原本プロセス生成から、プロセス生成を開始する。

（２）方式２（常駐対象：(B)、開始点：❸）

１回目のプログラム起動時に原本プロセスの常駐処理を行なう。２回目以降では、プロセス分割からプロセス生成を開始する。

（３）方式３（常駐対象：なし、開始点：❶❹）

１回目のプログラム起動時に、多数のコピープロセスを生成する。２回目以降では、以下の２通りの処理のどちらかが実行される。

(a) (生成プロセス数)≦(コピープロセスの数)

引数設定からプロセス生成を開始する。

(b) (生成プロセス数)＞(コピープロセスの数)

１回目の起動時と同様に、ディスクＩ／Ｏ処理か

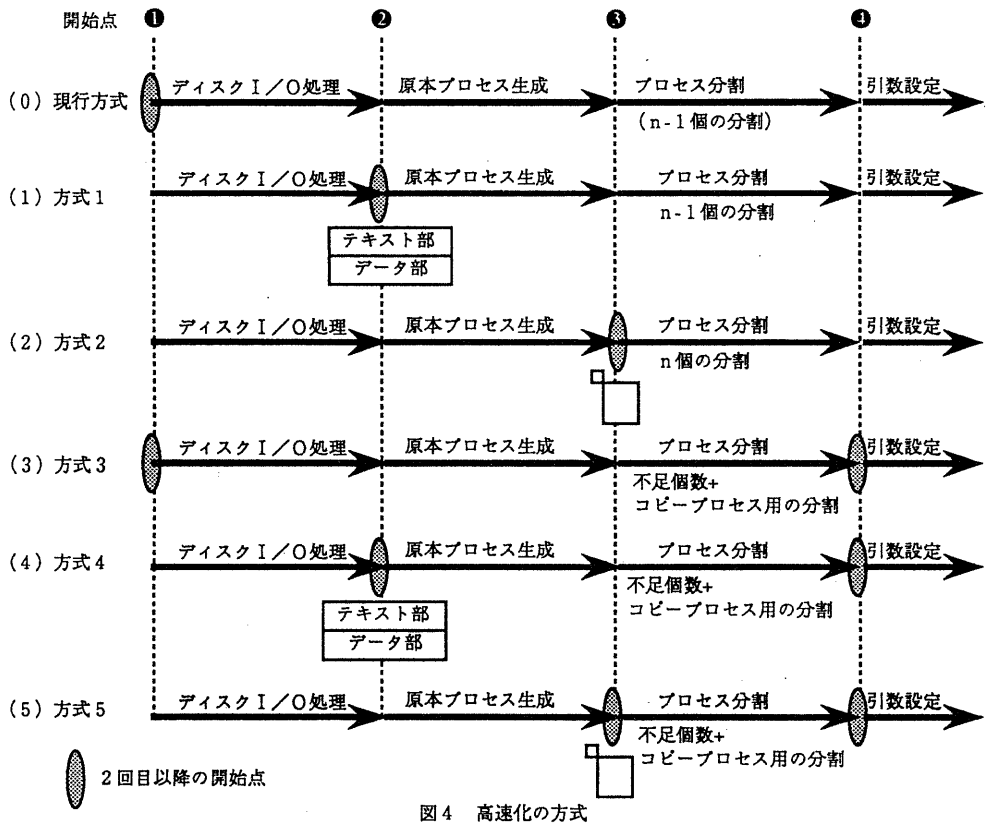

図4　高速化の方式

ら開始し、原本プロセス生成とプロセス分割を行なうことにより、多数のコピープロセスを生成する。

(4) 方式4 (常駐対象：(A)、開始点：❷❹)

1回目のプログラム起動時に、テキスト部とデータ部の常駐処理と、多数のコピープロセスを生成する。2回目以降では、以下の2通りの処理のどちらかが実行される。

(a) (生成プロセス数)≦(コピープロセスの数)
引数設定からプロセス生成を開始する。

(b) (生成プロセス数)＞(コピープロセスの数)
原本プロセス生成から開始し、プロセス分割を行なうことにより、コピープロセスを生成する。

(5) 方式5 (常駐対象：(B)、開始点：❸❹)

1回目のプログラム起動時に、原本プロセスの常駐処理と、多数のコピープロセスを生成する。2回目以降では、以下の2通りの処理のどちらかが実行される。

(a) (生成プロセス数)≦(コピープロセスの数)
引数設定からプロセス生成を開始する。

(b) (生成プロセス数)＞(コピープロセスの数)
プロセス分割から開始し、コピープロセスを生成する。

4.2.4　考察

各方式の比較を表3に示す。表3より、以下のことがわかる。

(1) メモリ量

2回目の開始点が後になるほど、多量になる。

(2) 1回目の生成処理

2回目の開始点が後になるほど、低速である。これは、2回目以降の生成処理を高速化するための処理が増加するからである。

(3) 2回目以降の生成処理

ディスクI／O処理の回数を削減するには、テキスト部とデータ部の常駐が有効である。また、開始点を

表3　高速化方式の比較

方式	メモリ量			生成処理の性能			
				1回目の生成処理		2回目以降の生成処理	
	常駐範囲	ｺﾋﾟｰﾌﾟﾛｾｽ	量	処理内容	速度	処理内容	速度
現行方式	無	無	無	ﾃﾞｨｽｸI/O処理： ﾃｷｽﾄ部：1回　ﾃﾞｰﾀ部：1回 ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：n回 管理ﾃｰﾌﾞﾙの初期化：n回 引数設定：n回	最も高速	ﾃﾞｨｽｸI/O処理 ﾃﾞｰﾀ部：1回 ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：n回 管理ﾃｰﾌﾞﾙの初期化：n回 引数設定：n回	最も低速
方式1	ﾃｷｽﾄ部 ﾃﾞｰﾀ部	無	少量	現行方式に加え、 常駐処理	高速	ﾒﾓﾘｺﾋﾟｰ処理 ﾃﾞｰﾀ部：n回 管理ﾃｰﾌﾞﾙの初期化：n回 引数設定：n回	やや低速
方式2	ﾃｷｽﾄ部 ﾃﾞｰﾀ部 ｽﾀｯｸ部 管理ﾃｰﾌﾞﾙ	無	やや少量	現行方式に加え、 ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：1回 管理ﾃｰﾌﾞﾙの初期化：1回 常駐処理	やや高速	ﾒﾓﾘｺﾋﾟｰ処理 ﾃﾞｰﾀ部：n回 管理ﾃｰﾌﾞﾙの初期化：n回 引数設定：n回	やや高速
方式3	無	有	やや多量	現行方式に加え、 ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：p回 管理ﾃｰﾌﾞﾙの初期化：p回	やや低速	(a) n≦pのとき 引数設定：n回 (b) n>pのとき 現行方式と同じ (生成ﾌﾟﾛｾｽ数>n−p)	低速
方式4	ﾃｷｽﾄ部 ﾃﾞｰﾀ部	有	多量	現行方式に加え、 ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：p回 管理ﾃｰﾌﾞﾙの初期化：p回 常駐処理	低速	(a) n≦pのとき 引数設定：n回 (b) n>pのとき 方式1と同じ (生成ﾌﾟﾛｾｽ数>n−p)	高速
方式5	ﾃｷｽﾄ部 ﾃﾞｰﾀ部 ｽﾀｯｸ部 管理ﾃｰﾌﾞﾙ	有	最も多量	現行方式に加え ﾒﾓﾘｺﾋﾟｰ処理： ﾃﾞｰﾀ部：p+1回 管理ﾃｰﾌﾞﾙの初期化：p+1回 常駐処理	最も低速	(a) n≦pのとき 引数設定：n回 (b) n>pのとき 方式2と同じ (生成ﾌﾟﾛｾｽ数>n−p)	最も高速

生成プロセス数：n、コピープロセス数：p

2ヶ所持つ方式（方式4や方式5）は高速であるが、コピープロセス数が生成プロセス数より少ない場合、コピープロセスを持たない方式（方式1や方式2）より遅い。

したがって、方式1と方式2がメモリ量と性能の両方に対して有効であると推察できる。

5．おわりに

並列度の動的な変化を考慮した、多数プロセス生成の高速化手法について述べた。今後は定量的評価を行なう予定である。

<参考文献>

[1] 村上,他:"ハイパー・スカラ・プロセッサ・アーキテクチャ", 並列処理シンポジウムJSPP'91 pp.133-140 (1991).

[2] 谷口,他:"分散型リアルタイムオペレーティングシステム：DIROS", 情処研報, 89-OS-45-9 (1989).

[3] 谷口:他："プロセス生成の高速化と並列化に関する検討", 信学報、 CPSY89-28 (1989).

[4] M. Accetta, et.al. : "Mach : A new kernel foundation for UNIX development", Proc. of USENIX Summer Conference, pp.93-112 (1989).

[5] 横山,他："軽いプロセスの生成終了処理に関する性能評価", 第43回情処全大, 4L-9 (1991).