

*Tender*における プロセス構成資源の事前生成による高速プロセス生成機能

田村 大¹ 山内 利宏¹ 谷口 秀夫¹

概要：プロセス生成処理は、メモリ空間の生成やプログラムの読み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、プロセス生成処理を高速化することが重要である。そこで、プロセス生成処理を高速化する手法として、*Tender* オペレーティングシステムにおける資源の分離と独立化に着目し、プロセス生成処理の一部を事前に行う手法について述べる。事前に生成した資源をプロセス生成時に使用することで、プロセス生成処理を高速化できる。本稿では、プロセスを構成する資源の事前生成による高速プロセス生成機能の設計を示す。また、評価を行い、資源の事前生成による高速プロセス生成機能の有効性について報告する。

1. はじめに

計算機による様々なサービス提供が進むにつれて、計算機上で走行するソフトウェアが増加している。一方、計算機ハードウェア、特に CPU の高性能化は著しい。このため、同じプログラム処理の実行時間は、大きく短縮されている。また、マルチコアプロセッサの普及により、搭載されるコア数が増加している。つまり、CPU 使用率は常に 90%を超えているわけではなく、実入出力終了待ちなどによる CPU アイドル状態も存在する。また、メモリ価格の低下により、計算機に搭載されるメモリ量は増加傾向にあり、未使用メモリ量も従来に比べ増加している。そこで、アイドル状態の計算機ハードウェアを有効に利用できれば、処理を高速化できる。例えば、アイドル状態の磁気ディスクを使用し、応用プログラム（以降、AP）の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 の処理を高速化する研究 [1][2] がある。このように、アイドル状態の計算機ハードウェアを有効に利用できれば、本来のオペレーティングシステム（以降、OS）の処理や AP の処理に影響を与えることなく、処理を高速化できる。

OS は、プログラムを実行するために、プロセスを生成し、実行し、削除する。プロセス生成処理は、メモリ空間の生成やプログラムの読み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、AP がプロセスの生成処理を頻繁に行う場合、AP がサービスを提

供するまでの待ち時間が増加することや、他の AP に影響を与えることが考えられる。したがって、プロセス生成処理の高速化が重要である。

Tender オペレーティングシステム [3] (The ENduring operating system for Distributed EnviRonment) (以降、*Tender*) は、資源を分離し独立化して管理している。このため、プロセスを構成する資源（以降、プロセス構成資源）は、プロセスを構成し利用されていない場合でも、存在できる。この特徴を生かし、プロセス構成資源を再利用することで、プロセス生成処理を高速化できることを示した [4][5]。

本稿では、プロセス構成資源を再利用するだけでなく、事前生成して利用することで、プロセス生成処理を高速化できる未使用資源管理機構を提案する。資源を事前生成する際は、アイドル状態の CPU と未使用メモリを使用する。

プロセス生成処理を高速化する研究として、UNIX[6] の sticky bit や vfork システムコールがある。また、プロセス生成処理を投機的に行う方法 [7] がある。しかし、我々の提案する方法は、プロセスの構成する資源を事前に生成し、必要時に使用するため、これらの方法とは異なる。

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では、OS が制御し管理する対象を資源として細分化し、資源の分離と独立化を行っている。Linux などの既存 OS は、資源に相互の依存関係があるため、OS が扱う資源の粒度が大きく、かつ資源を構成する資源は独立して存在できない。一方、*Tender* は、資源の分離と独立

¹ 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology,
Okayama University

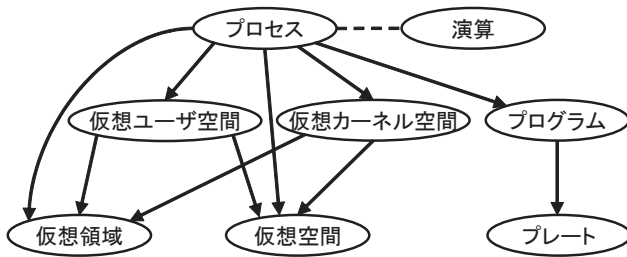


図1 Tenderにおけるプロセス構成資源の関係

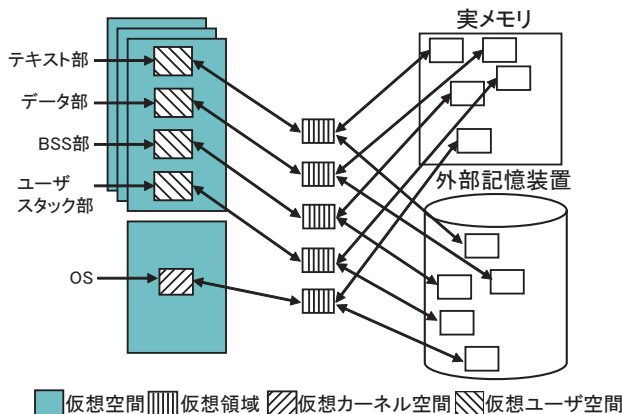


図2 Tenderにおけるメモリ関連資源の関係

化により、個々の資源が独立して存在できる。資源の分離と独立化を行うことで、各資源を操作するためのプログラムを部品化できる。このため、機能の追加や削除が容易になる。また、各資源を操作するためのプログラムの呼び出しは、特定のプログラムを介する特徴がある。このプログラムが各資源を管理するプログラムの呼び出しを管理するため、OSの動作や内部状態の理解や把握が容易になる。

2.2 プロセス構成資源

Tenderにおいて、プロセスを構成する資源を図1に示し、以下で説明する。図1において、矢印は、資源の依存関係を示しており、矢印の始点の資源は矢印が指す資源から構成される。「プロセス」は、プロセス識別子とプロセス管理表からなり、Tenderにおけるプログラムの実行単位である。「プログラム」は、実行プログラムのテキスト部とデータ部の大きさと先頭アドレス、および実行プログラムの開始アドレスの情報からなる。また、実行プログラムの内容は「プレート」が提供する。「プレート」は、永続的な記憶をメモリ上に提供する資源であり、既存OSのファイルに相当する。また、「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、プロセッサ割り当てを受けて走行できる。

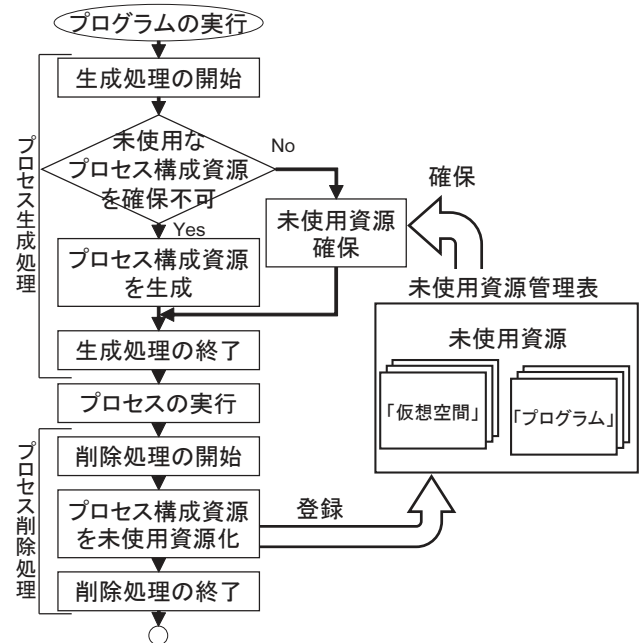


図3 プロセス構成資源の再利用機能

また、Tenderにおけるメモリ関連資源である「仮想空間」、「仮想領域」、「仮想ユーザ空間」、および「仮想カーネル空間」の関係を図2に示し、以下で説明する。「仮想空間」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。「仮想領域」は、実メモリと外部記憶装置の領域を仮想化した領域であり、サイズはページサイズ(4KB)の整数倍である。この領域の実体は、実メモリか外部記憶装置の領域上に存在する。「仮想カーネル空間」と「仮想ユーザ空間」は、「仮想領域」を「仮想空間」のもつ仮想アドレスと対応付けたものである。以降、この対応付けを仮想領域の貼り付けと呼び、対応付けを解除することを仮想領域の剥がしと呼ぶ。「仮想カーネル空間」と「仮想ユーザ空間」の差異は、ユーザモードで走行するプログラムがアクセス可能か否かの違いであり、「仮想ユーザ空間」は、ユーザモードとカーネルモードで走行するプログラムのどちらもアクセス可能である。これに対し、「仮想カーネル空間」は、カーネルモードで走行するプログラムのみアクセス可能である。

プロセスのテキスト部、データ部、BSS部、およびユーザスタック部は、「仮想ユーザ空間」として「仮想空間」上に存在する。ここでデータ部は、初期値をもつ変数の集合である。これに対してBSS部は、初期値をもたない変数の集合である。

2.3 プロセス構成資源の再利用機能

プロセス構成資源の再利用機能を図3に示し、以下で説明する。プログラムの実行により、OSはプロセスを生成し、実行し、削除する。Tenderにおいて、プロセス削除

表 1 再利用できる資源

通番	資源の種類	事前生成できる資源	利用可否の観点
1-A	プログラム 非依存資源	ワーク領域用仮想カーネル空間	常に利用可能
1-B		「仮想空間」	プログラム内容を意識した仮想空間がメモリ上にない場合、常に利用可能
1-C		「仮想領域」	サイズ
2-A	プログラム 依存資源	「プログラム」	プログラム内容
2-B		プログラム内容を意識したテキスト部用仮想領域	プログラム内容
2-C		プログラム内容を意識した仮想空間	プログラム内容

の際、再利用可能なプロセス構成資源を未使用な資源（以降、未使用資源）を管理する管理表（以降、未使用資源管理表）に登録し、実際には削除しないことにより、プロセス構成資源の削除にかかる処理時間を短縮し、高速なプロセス削除処理を実現している。また、プロセス生成処理の際、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、再利用可能なプロセス構成資源を確保し、プロセス生成処理を高速化している。

Tender におけるプロセス生成処理は、プログラムの内容を実メモリ上にすべて読み込む。このため、資源を再利用しない場合のプロセス生成処理にかかる処理時間は Linux などの既存 OS より長い。しかし、プロセス構成資源の再利用機能により、プロセス生成処理の時間を大きく短縮できる。また、実行時には、ページ例外が発生せず、Demand Paging や Copy on Write を実現している既存 OS と比べ、オーバーヘッドが小さい。

2.4 再利用できる資源

再利用できる資源は、特定のプログラムに依存するか否かにより 2 つに分類できる。1 つは、特定のプログラムに依存せず、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）である。もう 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）である。

再利用できる資源を表 1 に示し、以下で説明する。

(1) プログラム非依存資源

プログラム非依存資源として、ワーク領域用仮想カーネル空間、「仮想空間」、および「仮想領域」がある。

(A) ワーク領域用仮想カーネル空間

ワーク領域用仮想カーネル空間は、新規プロセスに渡す引数を一時的に格納するための「仮想カーネル空間」である。新規プロセスに渡すことができる引数のサイズは 4KB 以下であるため、ワーク領域用仮想カーネル空間のサイズは、

常にページサイズ（4KB）である。このため、ワーク領域用仮想カーネル空間を常に再利用可能である。

(B) 「仮想空間」

「仮想空間」は、プログラム内容を意識した仮想空間が再利用可能でない場合、常に再利用可能である。

(C) 「仮想領域」

「仮想領域」は、プログラムには依存しないがページサイズ（4KB）の整数倍で管理されている。このため、サイズが同じであれば、「仮想領域」を再利用可能である。

(2) プログラム依存資源

プログラム依存資源として「プログラム」、プログラム内容を意識したテキスト部用仮想領域、およびプログラム内容を意識した仮想空間がある。

(A) 「プログラム」

「プログラム」は、実行プログラムの情報をもつ。

(B) プログラム内容を意識したテキスト部用仮想領域

プログラム内容を意識したテキスト部用仮想領域は、対応するメモリ上に特定の実行プログラムのテキスト部のデータが存在する「仮想領域」である。

(C) プログラム内容を意識した仮想空間

プログラム内容を意識した仮想空間は、あるプログラムに対応するテキスト部、データ部、BSS 部、およびユーザスタック部用の仮想領域が貼り付けられている「仮想空間」である。

3. 未使用資源調整機構

3.1 プロセス構成資源の再利用機能の問題点

プロセス構成資源の再利用機能は、プロセス構成資源を再利用することによりプロセス生成処理と削除処理を高速化する。しかし、以下の問題点がある。

(問題点 1) 再利用可能な未使用資源の不足

プロセス構成資源の再利用機能は、プロセス削除時にプロセス構成資源をメモリ上に保持することにより実現している。このため、プロセス構成資源を必要としている際、プロセス構成資源が未使用資源としてメモリ上に必要な量だけ存在せず、処理を高速化できない場合がある。

(問題点 2) 再利用されない未使用資源の削除

プロセス構成資源の再利用機能は、プロセス構成資源を未使用資源管理表に登録し、必要時に使用する。しかし、未使用資源管理表に登録されている資源は、使用されなければいつまでもメモリ上に保持され続けることにより、メモリを無駄に使用する。このため、再利用されない未使用資源を削除する必要がある。

上記の問題点への対処として、未使用資源管理機構を提案する。未使用資源管理機構を図 4 に示し、以下で説明する。未使用資源管理機構は、プロセス構成資源の再利用機能、および事前生成と削除機能をもつ。未使用資源の事前生成と削除は、CPU のアイドル状態を使用し、未使用資

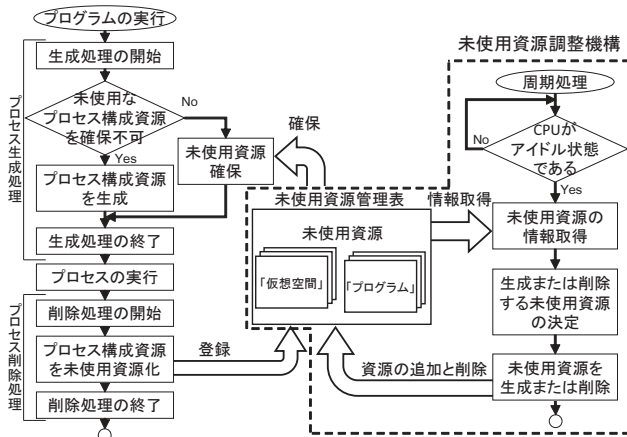


図4 未使用資源管理機構

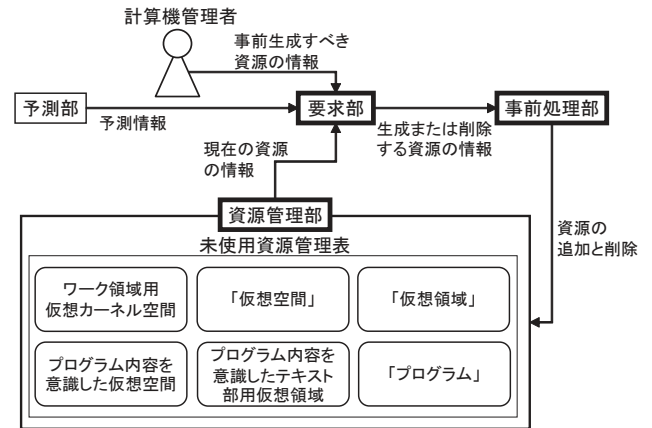


図5 未使用資源調整機構

源の量を調整する。具体的には、不足している未使用資源を生成し、使用されない未使用資源を削除する。これにより、(問題点1)と(問題点2)に対処できる。上記を実現するための機構として、未使用資源調整機構を提案する。

3.2 課題

未使用資源調整機構を実現するための検討課題として、以下の2つが考えられる。

(課題1) 事前生成する資源の種類と量

プログラム依存資源を事前生成する場合、対応するプログラムが実行されなければ、事前生成した資源がメモリ上に残り続ける。これにより、使用メモリ量が増加し、メモリ空間の確保といった本来のOSの処理に影響を与えることが考えられる。このため、使用する資源のみを事前生成する必要がある。

(課題2) 処理の契機

資源の事前生成の処理にはCPU時間を使用する。このため、事前生成の処理がCPUを占有し、本来のOSの処理とAPの処理に影響を与えることが考えられる。しかし、未使用資源調整機構の処理は補助的な機能であるため、本来のOSの処理とAPの処理に影響を与えることを抑制する必要がある。

(課題3) 処理の主体

未使用資源調整機構をユーザプログラムで実現するかカーネル内部に実現するかを検討する必要がある。

3.3 対処

3.2節で示した課題への対処を以下で示す。

(対処1) 事前生成できる資源は、2.4節で示した再利用できる資源と同じ資源を対象とする。また、事前生成と再利用によりメモリ上に保持する資源の種類と量は計算機管理者が設定する。計算機管理者は設定の際、事前生成により、本来のOSの処理とAPの処理に影響を与えないよう

に設定する。計算機管理者が設定する情報は、信頼できるものとする。これにより、(課題1)に対処できる。

(対処2) プロセス構成資源の生成処理は、アイドル状態のCPUを使用し、未使用資源を生成または削除する。これにより、(課題2)に対処できる。

(対処3) 未使用資源処理機構は、カーネル内部に実現する。これは、未使用資源の情報やCPUがアイドル状態であるかを判定するために使用する情報といったカーネル内部の情報にアクセスできるようにするためである。

3.4 設計

未使用資源調整機構を図5に示し、以下で説明する。未使用資源調整機構は、要求部、資源管理部、事前処理部、および予測部の4つの処理部で構成する。また、図5における矢印は、処理部間で受け渡しする情報を示している。それぞれの処理部を以下で説明する。

(1) 要求部

要求部は、事前生成する資源と未使用資源について、資源の種類と量の情報を取得する。情報を取得後、生成または削除する資源の種類と量を決定し、事前処理部に資源の生成または削除を要求する。

(2) 資源管理部

資源管理部は、未使用資源管理表で未使用資源を管理し、要求部からの要求に応じて、未使用資源の種類と量の情報を渡す。

(3) 事前処理部

事前処理部は、CPUのアイドル時間を使用し、要求部から要求された資源を生成または削除する。

(4) 予測部

予測部は、必要となると考えられる資源の種類と量を予測し、予測した情報を要求部に渡す。

以降では、要求部、資源管理部、および事前処理部の設計を示す。なお、予測部の実現は、今後の課題とする。

表 2 未使用資源の情報の管理形式

(情報 A) 資源の種類	(情報 B) プログラム名かサイズ	(情報 C) 資源数
ワーク領域用仮想カーネル空間	null	2
「仮想空間」	null	5
「仮想領域」	4 KB	5
	8 KB	1
	:	:
	(4 × n) KB	3
「プログラム」	prog_E	1
	prog_F	1
	:	:
プログラム内容を意識したテキスト部用仮想領域	prog_C	1
	prog_D	1
	:	:
プログラム内容を意識した仮想空間	prog_A	2
	prog_B	10
	:	:

3.5 要求部

3.5.1 未使用資源量と必要資源量の取得

要求部は、生成する未使用資源を決定するために、未使用資源量の情報と必要資源量の情報の 2 つを取得する必要がある。

未使用資源量の情報は、未使用資源の種類と量の情報である。この情報は、資源管理部から取得する。要求部が資源管理部から取得する未使用資源の情報の管理形式を表 2 に示す。(情報 A) 資源の種類は、事前生成できる資源の種類である。(情報 B) プログラム名かサイズは、プログラムまたはサイズに依存する資源を生成する際に指定する情報である。なお、プログラムに依存する資源は「プログラム」、プログラム内容を意識したテキスト部用仮想領域、およびプログラム内容を意識した仮想空間であり、サイズに依存する資源は、「仮想領域」である。(情報 C) 資源数は、未使用資源の数を指定する。

必要資源量の情報は、これから必要となる資源の種類と量の情報である。この情報は、計算機管理者が手動で設定する。必要資源量の情報は、表 2 の形式である。ただし、表 2 の(情報 C) に必要資源数を指定する。

上記の 2 つの情報を取得することで、実際の未使用資源の量と、事前生成すべき未使用資源の量を比較し、新たに生成すべき未使用資源の量を決定できる。

3.5.2 情報の解析と要求する処理の決定

3.5.1 項で示した未使用資源量と必要資源量をもとに、生成または削除する未使用資源の種類と量を決定する。各条件と処理を表 3 に示し、以下で説明する。

未使用資源量が必要資源量より少ない場合、事前処理部へ資源の生成を要求する。未使用資源量が必要資源量より多い場合、閾値を使用して処理を決定する。未使用資源量が閾値を超えていれば事前処理部へ資源の削除を要求し、

そうでなければ処理を行わない。閾値を設定することで、未使用資源量が大きくなった場合でも少しであれば許容し、削除する資源量を減らすことができる。なお、閾値の設定法は今後の課題である。未使用資源量と必要資源量が同じ場合、処理を要求しない。

3.5.3 生成または削除処理の要求

要求部が事前処理部へ処理を要求する際に、事前処理部へ与える情報の形式は表 2 の形式である。ただし、表 2 の(情報 C) に、生成または削除する資源数を指定する。生成または削除する資源数とは、要求部が決定した未使用資源の生成数または削除数である。正の数が生成数であり、負の数が削除数である。

3.6 資源管理部

資源管理部は、要求部から未使用資源量の取得要求を受け取り、未使用資源管理表を参照する。その後、未使用資源量の情報を要求部へ返却する。資源管理部が要求部に返却する情報の形式は表 2 の形式である。

3.7 事前処理部

事前処理部は、要求部の要求に基づき、CPU のアイドル時間を使用して、資源を生成または削除する。CPU がアイドル状態であるかの判定方法を表 4 に示し、以下で説明する。

事前処理部は、優先度が最低に設定されているアイドルプロセスが CPU を使用している時間を定期的に取得する。アイドルプロセスが CPU を使用している時間が閾値を超えた場合、CPU をアイドル状態であるとし、未使用資源の生成または削除処理を行う。なお、閾値の設定法は今後の課題である。この処理は、「演算」の各プロセスの実行時間を記録する機能を利用して実現する。

CPU がアイドル状態であるかを判定する処理は定期的に行われる。このため、アイドル状態であるかを判定するための処理が複雑な場合、判定のために多くの時間を CPU へ割り当てる必要がある。よって、アイドル状態であるかを判定するための処理は、簡単な処理である必要がある。表 4 に示した判定方法であれば、アイドルプロセスの走行時間を取得し、判定する処理のみで実現できる。このため、この判定方法を使用する。

4. 評価

4.1 評価内容と評価環境

プロセス構成資源の事前生成の有効性を明らかにするために、以下の評価を行った。

(評価 1) 事前生成なしの場合におけるプロセス生成処理にかかる処理時間

事前生成なしの場合におけるプロセス生成の処理時間を明らかにするために、プロセス生成処理を行うカーネルコー

表 3 生成または削除する未使用資源の種類と量の決定法

条件	処理
未使用資源量 < 必要資源量	未使用資源の生成処理
未使用資源量 > 必要資源量	未使用資源量 ≤ 閾値 未使用資源量 > 閾値
	処理なし 未使用資源の削除処理
未使用資源量 = 必要資源量	処理なし

表 4 CPU がアイドル状態であるかの判定方法

条件	事前処理部の処理
アイドルプロセスが CPU を使用している割合 > 閾値	CPU をアイドル状態とみなし、資源の生成または削除処理
アイドルプロセスが CPU を使用している割合 ≤ 閾値	CPU をアイドル状態ではないとみなし、処理なし

表 5 評価環境

OS	Tender
CPU	Celeron D (2.80GHz) (1 コア)
RAM	768 MB

ルの処理時間を測定する。

（評価 2）事前生成できる資源の確保にかかる処理時間
事前生成なしとありの場合において、事前生成できる資源の確保にかかる処理時間の差を明らかにする。このために、プロセス生成処理の際、事前生成できる資源の確保にかかる処理時間を事前生成なしとありの場合で測定する。

上記の評価において、処理時間を測定する際に使用するプログラムは、テキスト部、データ部、および BSS 部のサイズを可変で測定した。具体的には、サイズをそれぞれ、4KB, 16KB, 32KB, 64KB, 128KB, および 256KB と変え、測定した。また、事前生成機能の有効性を把握しやすくするため、ディスク I/O は発生しない場合とし、資源の再利用機能も使用しない場合とした。評価環境を表 5 に示す。なお、測定は rdtsc 命令を使用した。

4.2 評価結果

4.2.1 事前生成なしの場合におけるプロセス生成処理にかかる処理時間

事前生成なしの場合におけるプロセス生成処理を行うカーネルコールの処理流れを図 6 (1) に示す。また、事前生成なしの場合におけるプロセス生成処理を行うカーネルコールの処理時間 ($T_{proc_kerncall}$) を測定し、定式化したものを以下で示す。ここで、 X_{text} , X_{data} , および X_{BSS} はそれぞれテキスト部、データ部、および BSS 部のサイズ ($X_{text}KB$, $X_{data}KB$, および $X_{BSS}KB$) である。

$$T_{proc_kerncall} = 2.15 \times X_{text} + 2.12 \times X_{data} + 1.57 \times X_{BSS} + 393.1 \mu sec \quad (1)$$

4.2.2 事前生成できる資源の確保にかかる処理時間

事前生成ありの場合におけるプロセス生成処理を行うカーネルコールの処理流れを図 6 の (2) から (4) に示す。ここで、図 6 の (2) から (4) について、以下で説明する。

(2) プログラム非依存資源を事前生成

ワーク領域用仮想カーネル空間、「仮想空間」、および「仮想領域」（テキスト部、データ部、BSS 部、およびユーザスタック部用）を事前生成した場合のプロセス生成処理を行うカーネルコールの処理流れである。

(3) 部分的にプログラム内容を意識して資源を事前生成
ワーク領域用仮想カーネル空間、「プログラム」、「仮想空間」、プログラム内容を意識したテキスト部用仮想領域、および「仮想領域」（データ部、BSS 部、およびユーザスタック部用）を事前生成した場合のプロセス生成処理を行うカーネルコールの処理流れである。

(4) プログラム内容を意識して資源を事前生成
ワーク領域用仮想カーネル空間、「プログラム」、およびプログラム内容を意識した仮想空間を事前生成した場合のプロセス生成処理を行うカーネルコールの処理流れである。

また、事前生成なしとありの場合における事前生成できる資源の確保にかかる処理時間を表 6 に示し、以下で説明する。

(a) ワーク領域用仮想カーネル空間

ワーク領域用仮想カーネル空間の確保にかかる処理時間は、事前生成なしの場合、 $21.4 \mu sec$ であり、事前生成ありの場合、 $1.55 \mu sec$ である。よって、事前生成により処理時間を $19.85 \mu sec$ 短くできる。

(b) 「プログラム」

「プログラム」の確保にかかる処理時間は、事前生成なしの場合、 $6.73 \mu sec$ であり、事前生成ありの場合、 $1.08 \mu sec$ である。よって、事前生成により処理時間を $5.65 \mu sec$ 短くできる。

(c) 「仮想空間」

「仮想空間」の確保にかかる処理時間は、事前生成なしの場合、 $243.7 \mu sec$ であり、事前生成ありの場合、 $0.76 \mu sec$ である。よって、事前生成により処理時間を $242.94 \mu sec$ 短くできる。

(d) 「仮想領域」

「仮想領域」の確保にかかる処理時間は、事前生成なしの場合、 $0.23 \times X_{vr} + 5.00 \mu sec$ であり、事前生成ありの場合、 $0.56 \mu sec$ である。ここで、 X_{vr} は仮想領域のサイ



図 6 Tender におけるプロセス生成処理

表 6 事前生成できる資源の確保にかかる処理時間

通番	プロセス構成資源の種類	事前生成できる資源の確保にかかる処理時間		事前生成なしとありの場合の処理時間の差分
		事前生成なし	事前生成あり	
(a)	ワーク領域用仮想カーネル空間	$T_{work_vkm} = 21.4 \mu sec$	$T_{pre_work_vkm} = 1.55 \mu sec$	$19.85 \mu sec$
(b)	「プログラム」	$T_{prg} = 6.73 \mu sec$	$T_{pre_prg} = 1.08 \mu sec$	$5.65 \mu sec$
(c)	「仮想空間」	$T_{vm} = 243.7 \mu sec$	$T_{pre_vm} = 0.76 \mu sec$	$242.94 \mu sec$
(d)	「仮想領域」	$T_{vr} = 0.23 \times X_{vr} + 5.00 \mu sec$	$T_{pre_vr} = 0.56 \mu sec$	$0.23 \times X_{vr} + 4.44 \mu sec$
(e)	プログラム内容を意識したテキスト部用仮想領域	$T_{text_vr} = 1.83 \times X_{text} + 7.80 \mu sec$	$T_{pre_text_vr} = 0.13 \mu sec$	$1.83 \times X_{text} + 7.67 \mu sec$
(f)	プログラム内容を意識した仮想空間	$T_{prg_vm} = 2.15 \times X_{text} + 2.08 \times X_{data} + 1.45 \times X_{BSS} + 330 \mu sec$	$T_{pre_prg_vm} = 7.30 \mu sec$	$2.15 \times X_{text} + 2.08 \times X_{data} + 1.45 \times X_{BSS} + 322.7 \mu sec$

ズ (X_{vr} KB) である。なお、図 6 (1) の (処理 F) にかかる時間を計測した。また、(処理 H)、(処理 I)、および (処理 J) にかかる時間は、(処理 F) と同等の生成処理時間である。よって、事前生成により「仮想領域」の生成処理にかかる処理時間を $0.23 \times X_{vr} + 4.44 \mu sec$ 短くできる。

(e) プログラム内容を意識したテキスト部用仮想領域
プログラム内容を意識したテキスト部用仮想領域の確保にかかる処理時間は、事前生成なしの場合、 $1.83 \times X_{text} + 7.80 \mu sec$ であり、事前生成ありの場合、 $0.13 \mu sec$ である。よって、事前生成により処理時間を $1.83 \times X_{text} + 7.67 \mu sec$ 短くできる。

(f) プログラム内容を意識した仮想空間
プログラム内容を意識した仮想空間の確保にかかる処理時

間は、事前生成なしの場合、 $2.15 \times X_{text} + 2.08 \times X_{data} + 1.45 \times X_{BSS} + 330 \mu sec$ であり、事前生成ありの場合、 $7.30 \mu sec$ である。よって、事前生成により処理時間を $2.15 \times X_{text} + 2.08 \times X_{data} + 1.45 \times X_{BSS} + 322.7 \mu sec$ 短くできる。

4.2.3 資源の事前生成効果

資源の事前生成効果を図 7 に示し、以下で説明する。

(2) プログラム非依存資源を事前生成

(A) 事前生成により $0.23 \times (X_{text} + X_{data} + X_{BSS}) + 295.26 \mu sec$ の処理時間を短縮

(B) データ部と BSS 部が 4KB の場合で、プロセス生成処理にかかる時間を最大で 68.4%短縮

(3) 部分的にプログラム内容を意識して事前生成

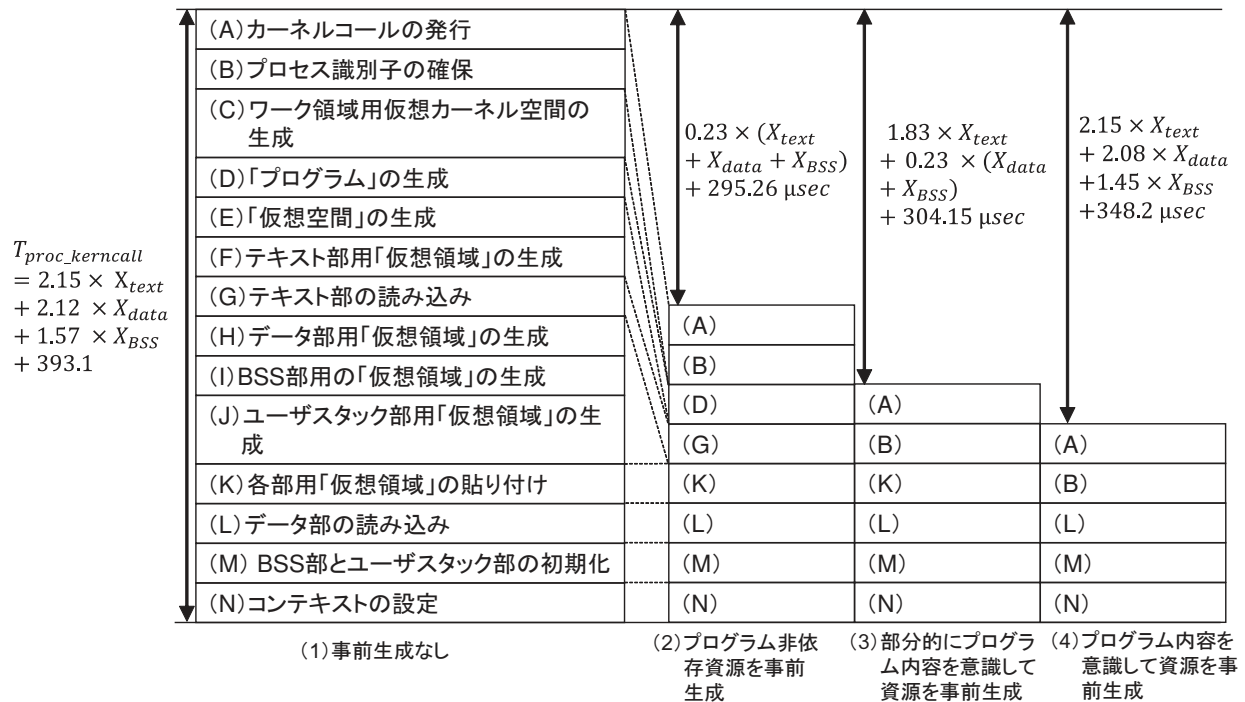


図 7 資源の事前生成効果

- (A) 事前生成により $1.83 \times X_{text} + 0.23 \times (X_{data} + X_{BSS}) + 304.15 \mu sec$ の処理時間を短縮
- (B) データ部と BSS 部が 4KB の場合で、プロセス生成処理にかかる時間の 75.3%以上の処理時間を短縮
- (4) プログラム内容を意識して事前生成
- (A) 事前生成により $2.15 \times X_{text} + 2.08 \times X_{data} + 1.45 \times X_{BSS} + 348.2 \mu sec$ の処理時間を短縮
- (B) データ部と BSS 部が 4KB の場合で、プロセス生成処理にかかる時間の 84.7%以上の処理時間を短縮

5. おわりに

プロセスを構成する資源の事前生成による高速プロセス生成機能について、設計と有効性を述べた。まず、プロセス生成処理の高速化の必要性について述べ、具体的な対処法として未使用資源管理機構を示した。未使用資源管理機構を実現するための機構として、未使用資源調整機構を示した。しかし、未使用資源調整機構の実現のためには検討すべき課題がある。そこで、課題と対処を示した。また、未使用資源調整機構のうち、要求部、資源管理部、および事前処理部の設計を示した。

評価では、プロセス生成の際の事前生成できる資源の確保にかかる処理時間を資源ごとに評価した。評価の結果、事前生成により、全ての事前生成できる資源の確保にかかる処理時間を短くできることを示した。特にプログラム内容を意識して資源を事前生成することにより、プロセス生成処理を 84.7%以上短くできることを示した。

残された課題として、閾値の設定法の検討と予測部の実現がある。

参考文献

- [1] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka: Informed Prefetching and Caching, 15th ACM Symposium on Operating Systems Principles, pp.79–95 (1995).
- [2] J. Lu, A. Das, W. Hsu, K. Nguyen: Dynamic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor, MICRO 38 Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture, pp.93–104 (2005).
- [3] 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).
- [4] 田端利宏, 谷口秀夫: 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方法の提案, 情報処理学会論文誌: コンピュータシステム, Vol.44, No.SIG10(ACS2), pp.48–61(2003).
- [5] T. Tabata, H. Taniguchi: An Improved Recyclable Resource Management Method for Fast Process Creation and Reduced Memory Consumption, International Journal of Hybrid Information Technology, Vol.1, No.1, pp.31–44 (2008).
- [6] J. S. Quarterman, A. Silberschatz, J. L. Peterson: 4.2BSD and 4.3BSD as examples of the UNIX system, ACM Computing Surveys, Vol.17, No.4, pp.379–418 (1985).
- [7] B. Wester, P. M. Chen, J. Flinn: Operating system support for application-specific speculation, EuroSys '11 Proceedings of the sixth conference on Computer systems, pp.229–242 (2011).