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の評価

田村 大[1]　佐藤 将也[1]　山内 利宏[1]　谷口 秀夫[1]

概要：プロセス生成処理は処理の負荷が大きいため，プロセス生成処理のオーバヘッドによって，応用プログラムやオペレーティングシステムの処理の性能が低下する可能性がある．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資源の事前生成機能について提案した．本稿では，資源の事前生成機能について，マイクロベンチマークと Apache Web サーバによる評価結果を報告する．

1. はじめに

計算機ハードウェアの性能向上と価格の低下に伴い，計算機ハードウェアが未使用である場合が増加している．具体的には，CPU がアイドル状態である時間の増加，搭載メモリ量の増加に伴う未使用なメモリ量の増加がある．したがって，アイドル状態の CPU と未使用メモリを用いて，処理の高速化が可能である．アイドル状態の計算機ハードウェアを使用し処理を高速化する研究として，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] がある．

オペレーティングシステム（以降，OS）はプログラム実行のためにプロセスを生成する．しかし，プロセス生成処理は，仮想アドレス空間の生成やプログラム読み込みを必要とするため，OS の処理の中でも負荷が大きい．このため，プロセス生成処理が AP の処理や他の OS の処理に影響を与える可能性があり，プロセス生成処理の高速化が必要である．

Tender オペレーティングシステム [2]（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，プロセスを構成し使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 [3]．また，プロセス構成資源を再利用するだけではなく，アイドル状態の CPU と未使用メモリを使用して，資源を事前生成し使用することで，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した [4]．

本稿では，未使用資源管理機構のうち，資源の事前生成機能について，マイクロベンチマークと Apache Web サーバを用いて，処理時間とメモリ使用量を評価した結果を報告する．

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

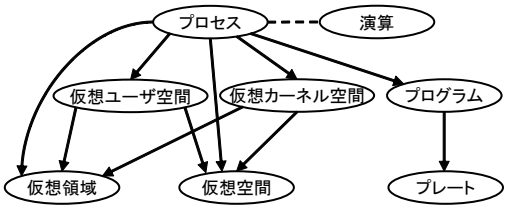

図 1　***Tender*** におけるプロセス構成資源の関係

2. *Tender* オペレーティングシステム

本章では，資源の事前生成機能を実現した ***Tender*** と評価に関係する ***Tender*** の機能について説明する．

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である資源を細分化し，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，資源の分離と独立化により，個々の資源が独立して存在できる．

2.2 プロセス構成資源

Tender において，プロセス構成資源の関係を図 **1** に示

© 2016 Information Processing Society of Japan

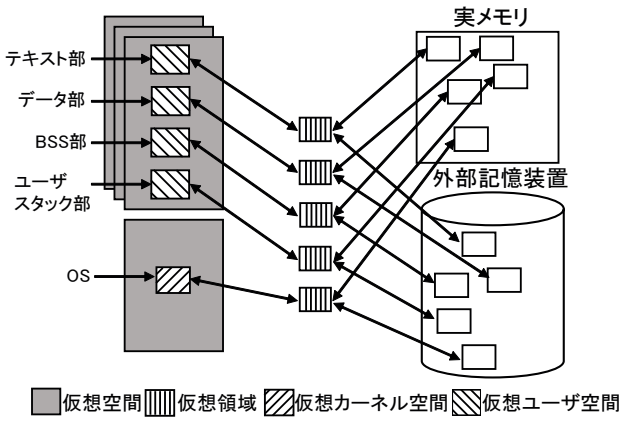

図 2 ***Tender*** におけるメモリ関連資源の関係

す．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．プロセス構成資源について説明する．「プロセス」は，プロセス識別子とプロセス管理表からなり，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サイズ，先頭アドレス，および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．「演算」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」に「演算」を割り当てることで「プロセス」はプロセッサ割り当てを受けることができる．

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」，「仮想カーネル空間」，および「仮想ユーザ空間」の関係を図 **2** に示す．「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．「仮想領域」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」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のに対し，「仮想ユーザ空間」は，ユーザモードとカーネルモードどちらの走行時も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り，BSS 部は，初期値をもたない変数の集合である．

2.3 資源の事前生成機能と再利用機能

Tender では，CPU のアイドル状態を使用し，必要となると考えられる資源を使用されていない資源（以降，未使

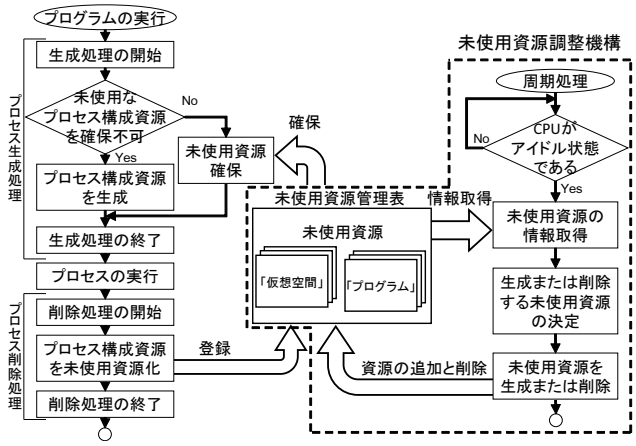

図 3 未使用資源管理機構

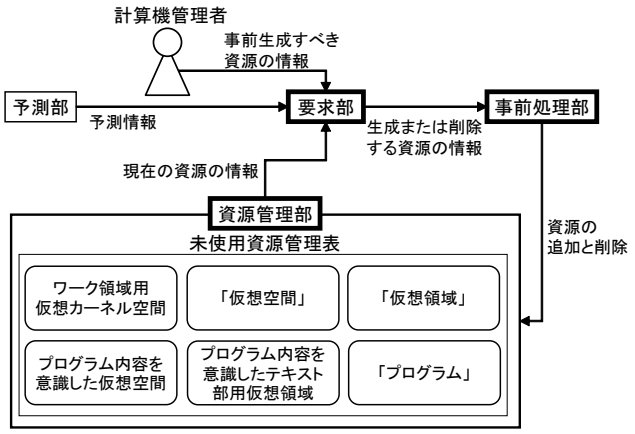

図 4 未使用資源調整機構

用資源）として生成しプロセス生成時に使用することで処理を高速化する資源の事前生成機能とプロセス削除時にプロセス構成資源を削除せず，未使用資源としてメモリ上に保持し，プロセス生成時に使用することで，高速にプロセスを生成できる資源の再利用機能がある．また，CPU のアイドル状態を使用し，使用されない未使用資源を削除することでメモリ使用量を減らす資源の自動削除機能がある．

未使用資源管理機構を図 **3** に示す．未使用資源管理機構では，資源の事前生成機能，資源の再利用機能，および資源の自動削除機能により未使用資源を管理する．資源の事前生成機能と資源の自動削除機能は未使用資源調整機構が処理を行う．なお，資源の事前生成機能と資源の再利用機能は実現しており，資源の自動削除機能の実現は今後の課題である．

2.4 未使用資源調整機構

未使用資源調整機構を図 **4** に示す．未使用資源調整機構は，資源管理部，要求部，事前処理部，および予測部の 4 つの処理部で構成される．

資源管理部は，未使用資源管理表で未使用資源を管理し，未使用資源の情報を要求部に渡す．要求部は，計算機管理

© 2016 Information Processing Society of Japan

表 1　事前生成または再利用できる資源

資源の種類	事前生成できる資源	使用可否の観点	メモリ使用量
プログラム非依存資源	ワーク領域用仮想カーネル空間	常に使用可能	4KB
	「仮想空間」	常に使用可能	$\boldsymbol{X_{pd}}$ = 4KB
	「仮想領域」	サイズが同じ場合	$\boldsymbol{X_{vr}}$ KB
プログラム依存資源	「プログラム」	プログラム内容	0KB
	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	$\boldsymbol{X_{text}}$ KB
	プログラム内容を意識した仮想空間	プログラム内容	$(\boldsymbol{X_{text} + X_{data} + X_{bss} + 64 + X_{pd} + X_{pts}})$ KB

者が設定する情報，資源管理部からの情報，および予測部からの情報から生成または削除する資源の種類と量を決定する．事前処理部は，要求部から要求された資源の生成または削除の処理を実行する．予測部は，必要となる資源を予測し，予測した情報を要求部に渡す．現在，要求部，資源管理部，および事前処理部を実現している．

2.5　事前生成または再利用できる資源

事前生成または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，使用できる資源（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，使用できる資源（以降，プログラム依存資源）である．事前生成または再利用できる資源を**表 1** に示す．

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．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．ワーク領域用仮想カーネル空間は，プロセス生成処理において必ず使用し，サイズが 4KB であるという観点から再利用あり，なしに関わらず，常に再利用する．「仮想空間」は，ページディレクトリ用のメモリ領域（$\boldsymbol{X_{pd}}$ KB）として，4KB 使用しており，常に使用できる．「仮想領域」は，ページサイズ（4KB）の整数倍で管理されている．このため，サイズが同じ場合，「仮想領域」を使用できる．仮想領域のサイズ（$\boldsymbol{X_{vr}}$ KB）に応じてメモリを使用する．

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．「プログラム」は，実行プログラムについて，テキスト部とデータ部のサイズと先頭アドレス，および開始アドレスの情報をもつ資源であり，***Tender*** 起動時に確保するメモリ領域を使用するため，動的にメモリを確保しない．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」であり，プログラムのテキスト部のサイズ（$\boldsymbol{X_{text}}$ KB）に応じてメモリを使用する．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」であり，プログラムのテキスト部（$\boldsymbol{X_{text}}$ KB），データ部（$\boldsymbol{X_{data}}$ KB），BSS 部（$\boldsymbol{X_{bss}}$ KB），ユーザスタック部（64 KB），ページディレクトリ（$\boldsymbol{X_{pd}}$ KB），およびページテーブル（$\boldsymbol{X_{pts}}$ KB）分メモリを使用する．

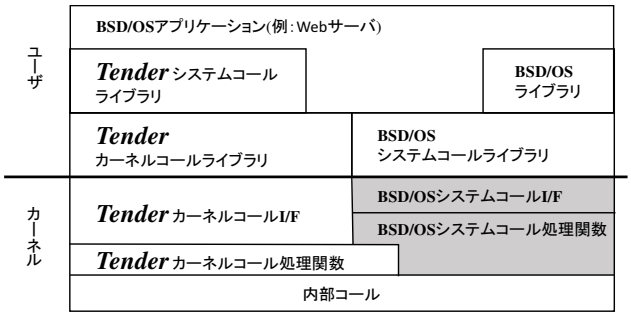

図 5　BSD/OS 互換システムコール I/F

2.6　BSD/OS 互換システムコール I/F

Apache Web サーバを ***Tender*** で実行するために実現した BSD/OS 互換システムコール I/F[5] について説明する．BSD/OS 互換システムコール I/F を**図 5** に示す．***Tender*** と BSD/OS は異なるコールゲートを用いており，共存させてもそれぞれの処理要求をカーネル内で受けとりできる．BSD/OS システムコール処理関数では，発行された BSD/OS のシステムコールに相当する処理を実行し，戻り値を返す．

BSD/OS 互換 I/F の fork と execve は，***Tender*** のプロセス変身機能 [6] を用いて実現されている．プロセス変身機能について説明する．プロセス変身機能は，実行プログラム変更機能，開始位置変更機能，および動作空間変更機能の 3 つの機能で構成される．実行プログラム変更機能を**図 6** に，開始位置変更機能を**図 7** に，動作空間変更機能を**図 8** に示す．実行プログラム変更機能は，プロセスとして実行されるプログラムを変更する機能である．開始位置変更機能は，対象のプロセスが次に走行する開始位置を変更する機能である．動作空間変更機能は，プロセスが利用

© 2016 Information Processing Society of Japan

表 2　fork と execve で使用可能な未使用資源

システムコール	fork	execve
使用可能な未使用資源	「仮想空間」,「仮想領域」,「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	「仮想領域」,「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

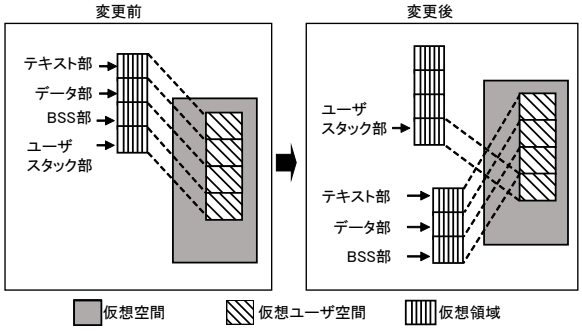

図 6　実行プログラム変更機能

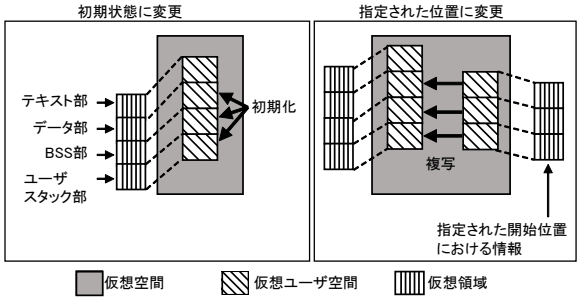

図 7　開始位置変更機能

する「仮想空間」を別の「仮想空間」に変更する機能である．これらの機能は，組み合わせて利用できる．また，プロセス変身機能では，未使用資源を使用し，処理を高速化できる．

表 2 に fork と execve で使用可能な未使用資源を示し，以下でそれぞれの処理を説明する．

fork は，動作空間変更機能，実行プログラム変更機能，および開始位置変更機能を使用する．まず，動作空間変更機能で子プロセスとして生成したカーネルプロセスに「仮想空間」を割り当て，実行プログラム変更機能で「仮想領域」の生成と「仮想ユーザ空間」の生成を行う．次に，開始位置変更機能により，親プロセスの状態を複写し，最後に「演算」を割り当てる．

execve は，実行プログラムの変更機能を使用し，execve を実行したプロセスの「仮想領域」と「仮想ユーザ空間」を削除し，新たに実行されるプログラムの「仮想領域」と「仮想ユーザ空間」を生成する．

3. 評価

3.1 評価の観点

資源の再利用機能と比較し，資源の事前生成機能が有効である場合として，違うサイズのプログラムを複数実行する場合がある．そこで，基本評価として，違うサイズのプログラムを複数実行する際の処理時間とメモリ使用量について，事前生成と再利用の有無を組み合わせた 4 つの場合でそれぞれ測定する．また，資源の再利用機能は，再利用できる資源が不足する場合がある．そこで，Web サーバを使用し，事前生成と再利用の有無を組み合わせた 4 つの場合における Web サーバの応答時間とメモリ使用量を測定し，考察する．

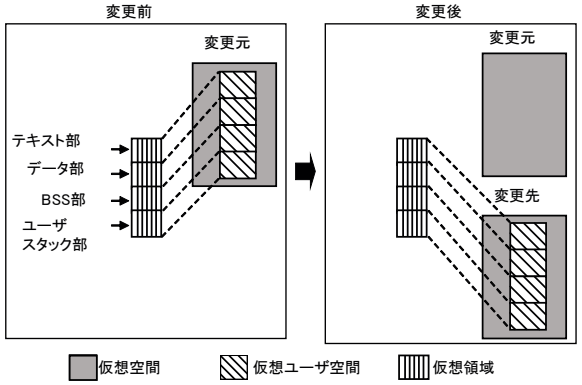

図 8　動作空間変更機能

3.2 基本評価

3.2.1 評価内容

基本評価として，**表 3** で示すテキスト部，データ部，BSS 部のサイズが 4KB の整数倍である 50 個のプログラムからプロセス生成処理と削除処理を 1 回ずつ行う処理にかかる処理時間の合計を測定する．また，プロセス削除時に未使用資源のメモリ使用量も測定する．プロセス生成の順番は，サイズが小さい順，サイズが大きい順，ランダムの 3 つの場合で行う．また，測定の条件として，事前生成ありの場合，「仮想空間」,「仮想領域」,「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を過不足なく事前生成する．再利用ありの場合，「仮想空間」,「仮想領域」,「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を再利用する．

計算機は，CPU: Celeron D (2.80 GHz)(1 コア)，RAM : 768 MB のものを使用した．また，資源の事前生成機能の有効性を明らかにするため，ディスク I/O の発生しない環境で評価した．なお，測定には rdtsc 命令を使用した．

3.2.2 評価結果

プロセス生成と削除処理にかかる処理時間を**図 9**，未使用資源のメモリ使用量を**図 10** に示す．なお，プログラムの実行順による測定結果の違いは見られなかったため，サイズが小さい順の測定結果のみを示す．図 9 より，以下の

© 2016 Information Processing Society of Japan

表 3　プロセス生成処理を行う際に使用する 50 個のプログラム

通番	プログラム名	テキスト部	データ部	BSS 部
1	prog4	4 KB	4 KB	4 KB
2	prog8	8 KB	8 KB	8 KB
:	:	:	:	:
n	prog4×n	4×n KB	4×n KB	4×n KB
:	:	:	:	:
50	prog200	200 KB	200 KB	200 KB

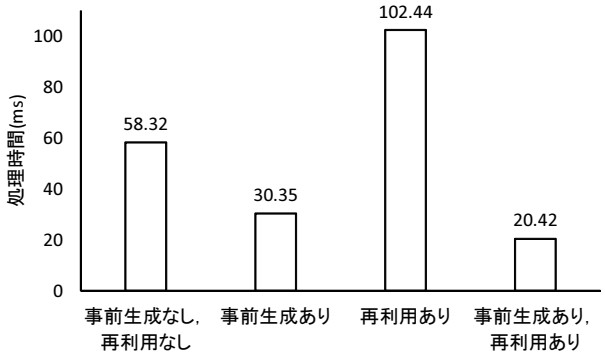

図 9　プロセス生成と削除にかかる処理時間

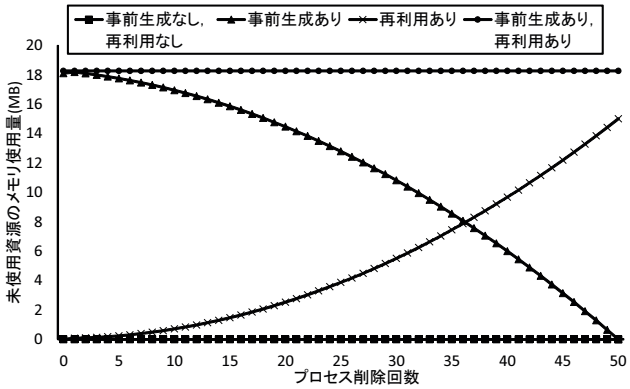

図 10　未使用資源のメモリ使用量

ことが分かる．

(1) 事前生成ありの場合，処理時間を約 27.9 ms（=58.3−30.4）（約 47.9%）短縮できる．これは，プロセス生成処理時において，事前生成した資源を使用することで，プロセス構成資源の生成処理を省略できるためである．

(2) 再利用ありの場合，処理時間が約 44.1 ms（=58.3−102.4）（約 75.6%）長くなる．これは，生成するプロセスの各部のサイズが異なるため，「仮想領域」とプログラム依存資源を再利用できないこと，未使用資源のメモリ使用量の増加によるメモリ確保のためのオーバヘッドの増加が原因である．

(3) 事前生成あり，再利用ありの場合，処理時間を約 37.9 ms（=58.3−20.4）（約 65.0%）短縮できる．これは，資源の事前生成機能によりプロセス構成資源の生成処理を省略できることに加え，資源の再利用機能によりプロセス削除時のプロセス構成資源の削除処理を省略できるためである．

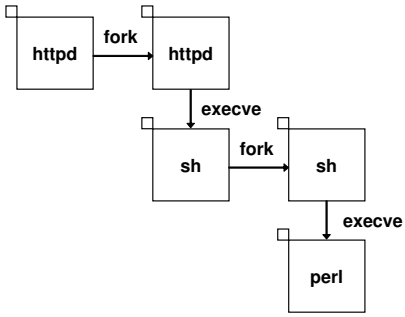

図 11　CGI プログラム実行時の処理流れ

また，図 10 より，再利用ありの場合，異なるサイズのプログラムからプロセスが生成されると未使用資源のメモリ使用量が大きくなり続ける．また，資源を事前生成する場合も，事前生成した資源が使用されないとメモリ使用量が大きくなると考えられる．このため，使用されないと考えられる未使用資源を削除する必要がある．

3.3　Web サーバによる評価

3.3.1　評価内容

Web サーバプログラムである Apache を使用し，Web サーバがプロセス生成を行う事例として，perl で作成した CGI プログラムの実行を伴う処理を Web サーバに要求し，その応答時間を評価する．

CGI プログラム実行時の処理流れを図 **11** に示す．CGI プログラムは，Web サーバが sh を呼び出し，sh が perl 処理系を呼び出すことで実行される．つまり，1 回の要求で 2 個のプロセスが生成される．本評価で使用する CGI プログラムは，ファイルから値を読み出し，その値を表示する．

評価環境は，サーバ計算機（CPU: Celeron D (2.80 GHz)(1 コア)，RAM：768 MB）1 台とクライアント計算機（CPU: Intel(R) Core i7-4790S (3.20 GHz)(4 コア)，RAM：8,192MB）1 台を 100Base-TX の Ethernet で接続して行った．Web サーバプログラムとして Apache ver.1.3.33 を使用した．Apache は起動時に子プロセスを 1 個生成する．また，測定用の Web クライアントプログラムとして Web サーバの 1 要求あたりの応答時間を rdtsc 命令により測定するプログラムを使用した．測定用の Web クライアントプログラムは，子プロセス（以降，クライアントプロセス）を生成し，クライアントプロセスは Web サーバへの要求を 10 回行う．測定の際は，クライアントプロセスの数が 1 個，5 個，および 10 個の場合において測定した．また，資源の事前生成機能の有効性を明らかにするため，ディスク I/O は発生しない場合とした．

httpd，sh，および perl のプログラムサイズを表 **4** に示す．事前生成ありの場合，それぞれ httpd が 15 回，perl と sh が 30 回プロセス生成された際に使用される「仮想空間」，「仮想領域」（データ部，BSS 部，ユーザスタック部用），「プログラム」，プログラム内容を意識したテキスト部

© 2016 Information Processing Society of Japan

表 4　プログラムサイズ

プログラム名	テキスト部	データ部	BSS 部
httpd	656 KB	36 KB	316 KB
sh	328 KB	16 KB	16 KB
perl	816 KB	40 KB	28 KB

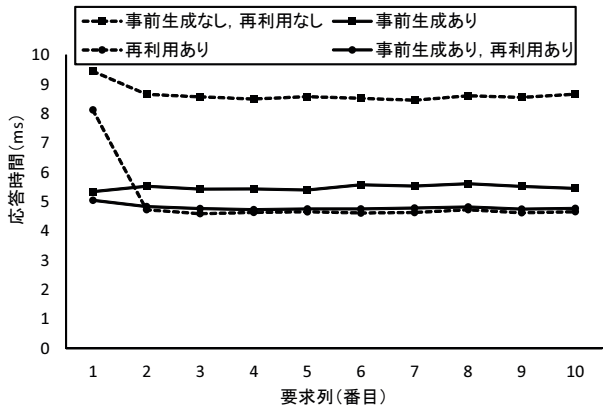

図 12　Web サーバの応答時間（クライアントプロセス 1 個）

用仮想領域を事前生成する．再利用ありの場合，「仮想空間」，「仮想領域」，「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を再利用する．

3.3.2　応答時間

クライアントプロセスが 1 個，5 個，および 10 個の場合における Web サーバの応答時間をそれぞれ**図 12**，**図 13**，および**図 14** に示す．

クライアントプロセスが 1 個の場合（図 12）における 1 回目の要求に対する応答時間は，事前生成ありの場合は約 5.3 ms であるのに対し，再利用ありの場合は約 8.1 ms と約 2.8 ms 長い．これは，事前生成ありの場合，未使用資源がメモリ上に保持されており，sh と perl のプロセス生成処理を高速化できたためである．これに対し，2 回目以降の応答時間においては，事前生成ありの場合は約 5.4 ms であるのに対し，再利用ありの場合は約 4.6 ms と事前生成ありの場合の方が約 0.8 ms 長い．これは，再利用ありの場合において，sh と perl のプロセス削除時にプロセス構成資源を削除せずメモリ上に保持するため，事前生成ありの場合と比べて高速にプロセスを削除できるためである．

また，図 13 と図 14 から，以下のことが分かる．

(1) 1 回目の要求に対する応答時間だけではなく，2 回目と 3 回目の要求に対する応答時間においても事前生成あり（図 13（B）と図 14（B））の方が再利用あり（図 13（C）と図 14（C））の場合より短い．これは，クライアントプロセスの要求が一度に複数生じたことにより Apache が子プロセスを複数生成したためである．子プロセスが生成されると再利用できる資源が減り，不足した資源を生成する処理が必要となるため，perl と sh のプロセス生成処理に時間がかかり，応答時間が長くなる．また，応答時間の平均も同様に，1〜3 回目の要求は事前生成ありの場合の方が再利用ありの場合より短い．4 回目以降の応答時間の平均は，クライアントプロセスが 5 個と 10 個の場合において，事前生成なし，再利用なし（図 13（A）と図 14（A））と比較し，事前生成あり（図 13（B）と図 14（B））の場合約 11.5 ms（=38.2−26.7）と約 18.6 ms（=74.3−55.7），事前生成あり，再利用あり（図 13（C）と図 14（C））の場合約 16.5 ms（=38.2−21.7）と約 32.0 ms（=74.3−42.3）小さくなる．

(2) 事前生成あり（図 13（B）と図 14（B））の場合，応答時間の最大値と最小値の差は最大約 55.5（=106.0−50.5）ms と小さい．また，事前生成あり，再利用あり（図 13（D）と図 14（D））の場合，差分は約 43.6（=85.9−42.3） ms となり，更に差分が小さい．これに対し，事前生成なし，再利用なし（図 13（A）と図 14（A））の場合，応答時間の最大値と最小値の差分は最大約 99.3（=184.1−84.8） ms，再利用あり（図 13（C）と図 14（C））の場合，最大約 76.2（=148.1−71.9） ms と大きい．最大値と最小値の差分が小さくなることにより，クライアントに提供するサービスの質の差を小さくできる．

3.3.3　メモリ使用量

Web サーバへの要求開始時から終了時における未使用資源のメモリ使用量を**図 15** に示す．測定は，未使用資源の量が変化した時点を全て測定しており，図 15 におけるマークの位置は測定時点の一定間隔ごとである．事前生成ありの場合，クライアントプロセスからのアクセス開始時に約 21.7MB メモリを使用している．これに対し，再利用ありの場合，要求開始時のメモリ使用量は 0MB である．しかし，クライアントプロセスが 5 個と 10 個の場合における要求終了時では，再利用ありの場合の方が事前生成ありの場合と比較し，未使用資源のメモリ使用量がそれぞれ約 9.4MB（=10.8−1.4），約 2.3MB（=5.6−3.3）大きい．また，事前生成あり，再利用ありの場合，未使用資源が常に約 18〜21MB メモリを使用している．このことから，資源を再利用する場合，事前生成する資源の量を減らすことができると考えられる．また，使用されない未使用資源を自動で削除する資源の自動削除機能の実現が必要である．

4.　関連研究

プロセス生成処理を高速化する手法として，UNIX [7] の Demand Paging や Copy on Write [8] がある．Demand Paging では，プロセス生成処理におけるプログラム読み込み処理の一部を遅延させることで，プロセス生成処理時間を短縮している．また，子プロセス生成時に Copy on Write を用いることによって，親プロセスと子プロセスの物理メモリを共有し，更新されたメモリのみを複写することでメモリ間のデータ複写を最小限にしている．

文献 [9] では，同じプログラムから複数のプロセスを生成する際にプログラムの読み込み回数を減らすことでプロ

© 2016 Information Processing Society of Japan

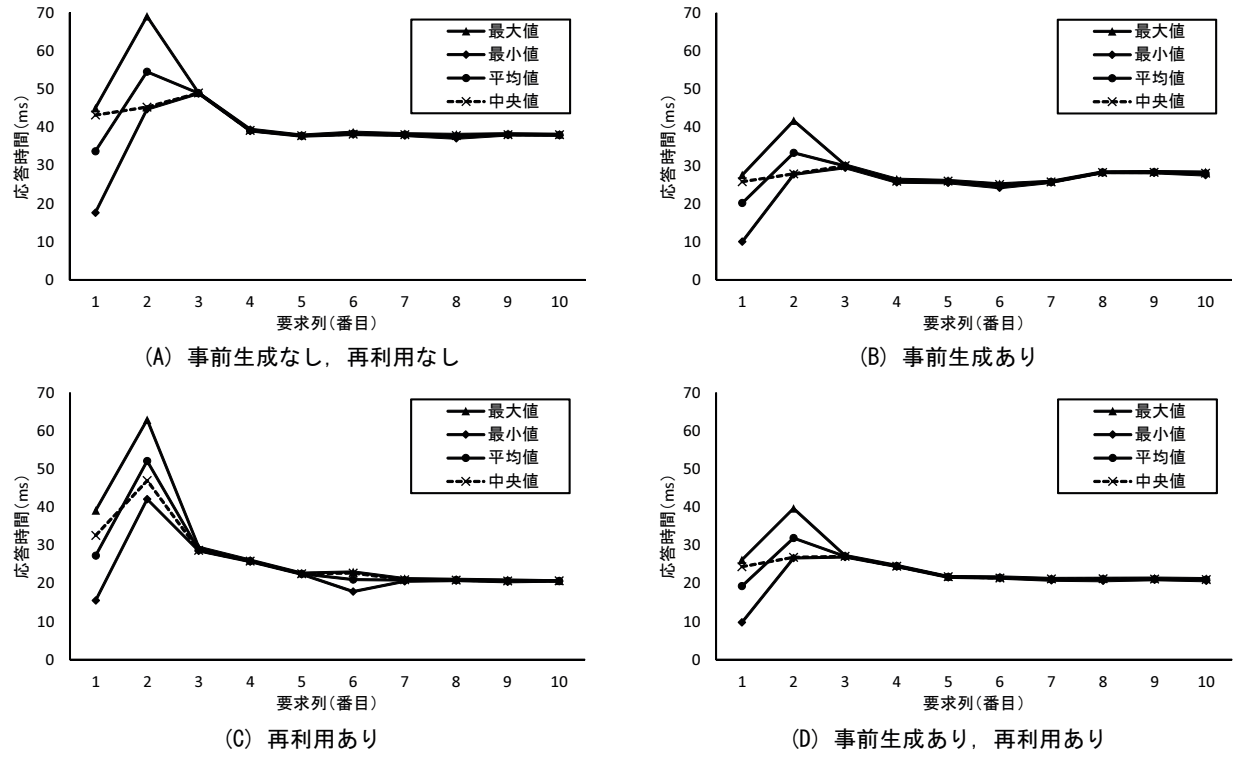

図 **13** Web サーバの応答時間（クライアントプロセス 5 個）

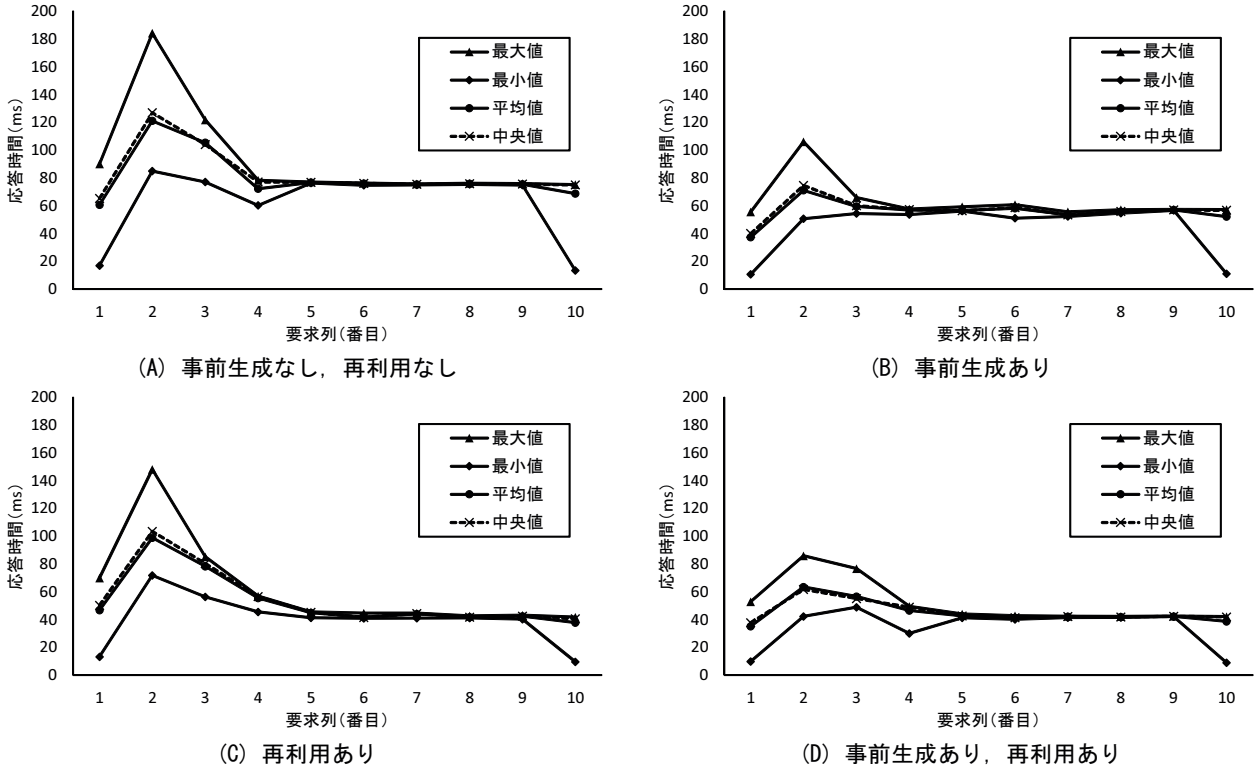

図 **14** Web サーバの応答時間（クライアントプロセス 10 個）

セス生成処理を高速化している．文献 [10] では，投機的にプログラムを実行し，プログラムの起動処理を高速化している．文献 [11] では，動的にリンクする共有ライブラリを事前に読み込むことで，そのライブラリを利用するプログラムの起動処理を高速化している．しかし，これらの手法は特定の条件やプログラムの改変が必要である．提案手法は，プロセスを構成する資源を事前に生成し，必要時に使用する方法であり，これらの手法より適用範囲が広く，プ

© 2016 Information Processing Society of Japan

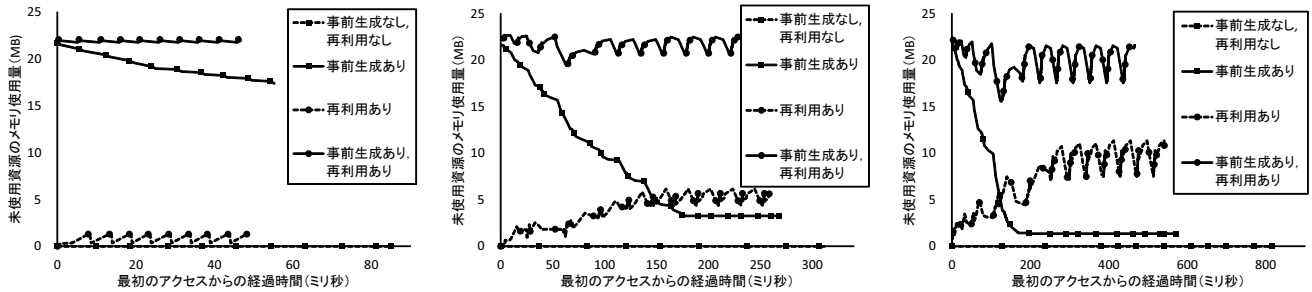

図 15　未使用資源のメモリ使用量

ログラムの改変も必要としない．

Web サーバへのアクセスにおいて CGI プログラムの実行を伴う場合，CGI の実行時に発生するプロセス生成の処理時間が問題となる．この問題解決のために SpeedyCGI [12]，FastCGI [13]，mod_perl [14] 等の手法がある．これらの手法は，プロセスの常駐化や Web サーバへのモジュール組込により CGI の実行を高速化する．しかし，これらの手法は Apache のソースコード改変や CGI プログラム改変が必要であり，適用されていない場合は，従来の fork&exec 処理により CGI を実行する．提案手法は，AP の変更を必要としない点と OS 上のすべての AP に適用できるという点で，AP レベルの対策手法よりも適用範囲が広い．

5. おわりに

本稿では，OS が制御し管理する対象である資源を細分化し資源の分離と独立化を行っている ***Tender*** において，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の評価を述べた

基本評価として，異なるサイズのプログラム 50 個からプロセス生成と削除を行い，その際の処理時間と未使用資源のメモリ使用量を測定した．測定結果より，資源を再利用するだけではプロセス生成処理を高速化できなかった．しかし，資源を事前生成することで約 27.9 ms（約 47.9%），資源の事前生成と再利用を組み合わせることで約 37.9 ms（約 65.0%）処理時間を短縮できることを示した．

また，Web サーバプログラムの Apache を使用し，Web サーバがプロセス生成を行う事例として，CGI プログラムの実行を伴う処理の応答時間を測定した．測定結果より，Web サーバへの初回の要求時や Apache の子プロセスが生成された直後といった再利用できる資源がメモリ上にない場合において，資源を事前生成することにより応答時間を短くできることを示した．また，Web サーバにおける未使用資源のメモリ使用量を測定した．測定の結果，事前生成機能も再利用機能も使用されない資源をメモリ上に保持することにより，メモリ使用量が増加することが明らかになった．この問題には，資源の自動削除が有効である．

残された課題として，資源の自動削除機能の実現，予測部の実現，マルチコア対応がある．

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] 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，” 情処学論，Vol.41，No.12，pp.3363–3374（2000）．

[3] 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，” 情処学論，Vol.44，No.SIG10(ACS2)，pp.48–61（2003）．

[4] 田村大，山内利宏，谷口秀夫，“***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，” 情処研報, Vol.2016-OS-137, No.10, pp.1–8 （2016）．

[5] 佐伯顕治，田端利宏，谷口秀夫，“***Tender*** の資源再利用機能を利用した高速 fork & exec 処理の実現と評価，” 情処学論，Vol.J91-D，No.12，pp.2892–2903（2008）．

[6] 石井陽介，田端利宏，谷口秀夫，“位置透過に利用可能な構成要素を用いたプロセス変身機能，” 情処学論，Vol.44，No.7，pp.1666–1679（2003）．

[7] 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).

[8] J. M. Smith, and G. Q. Maguire Jr., “Effects of copy-on-write memory management on the response time of UNIX fork operations,” COMPUTING SYSTEMS, Vol.1, No.3, pp.255–278 (1988).

[9] A. Kulkarni , A. Lumsdaine , M. Lang , L. Ionkov, “Optimizing latency and throughput for spawning processes on massively multicore processors,” ROSS ’12 Proc. 2nd International Workshop on Runtime and Operating Systems for Supercomputers, Vol.6 (2012).

[10] B. Wester, P. M. Chen, J. Flinn, “Operating system support for application-specific speculation,” Proc. sixth conference on Computer systems (EuroSys ’11), pp.229–242 (2011).

[11] C. Jung , D. Woo , K. Kim , S. Lim, “Performance characterization of prelinking and preloading for embedded systems,” Proc. 7th ACM & IEEE international conference on Embedded software, pp.213–220 (2007).

[12] S.Horrocks, “SpeedyCGI,” http://www.daemoninc.com/SpeedyCGI/, accessed Nov. 1 (2016).

[13] “FastCGI,” https://httpd.apache.org/mod_fcgid/, accessed Nov. 1 (2016).

[14] “mod_perl,” https://perl.apache.org/, accessed Nov. 1 (2016).

© 2016 Information Processing Society of Japan