Vol.2013-OS-124 No.11
2013/3/1

AnT オペレーティングシステムにおける 低機能MMUの制御法

鶴谷 昌弘[1]　山内 利宏[1]　谷口 秀夫[1]

概要：計算機の多様な利用を支える高い適応性と堅牢性を実現できる OS が必要となっており，これを実現する OS プログラム構造としてマイクロカーネル構造がある．マイクロカーネル OS は，OS 機能の大半を OS サーバとして実現するため，OS サーバ間でプログラム間通信が頻発し，モノリシックカーネル OS に比べ性能が低下する．このため，データ複写レスによる通信により，OS サーバ間での授受データの複写オーバヘッドを低減している．しかし，低機能 MMU では，データ複写レスであっても通信時に発生する TLB ミスに伴う処理オーバヘッドが大きい．そこで，マイクロカーネル OS における低機能 MMU 制御法を提案する．提案制御法は，サーバプログラム間通信で利用する領域についてはページテーブルを利用することなく，TLB エントリでページの割り当てを管理し，TLB ミスを発生させないことによりサーバプログラム間通信を高速化する．SH-4 を例として，提案制御法を ***AnT*** オペレーティングシステムに実現する方式を示し，性能評価の結果を報告する．

1. はじめに

計算機の多様な利用を支える高い適応性，および高い堅牢性を実現できる OS が必要となっている．これを実現する OS プログラム構造として，マイクロカーネル構造がある [1]−[3]．マイクロカーネル構造は，例外処理や割込処理といった最小限の OS 機能をカーネルとして実現し，ファイル管理やデバイスドライバなどの OS 機能をプロセス（OS サーバ）として実現するプログラム構造である．これにより，全 OS 機能をカーネルとして実現するモノリシックカーネル構造に比べ，機能の追加や削除を容易にできる．また，OS サーバごとに機能を分担させることにより，プログラムの暴走によるシステム全体の破壊を防止できる．近年，組込みシステムでは，プログラムの大規模化に伴う信頼性の低下が問題となっており [4]，マイクロカーネル構造が有用だと考えられる．

しかし，マイクロカーネル構造では，OS サーバ間でプログラム間通信が頻発するため，モノリシックカーネル構造の OS に比べ，性能が低下する．そこで，この性能低下を抑制する機構が必要となる．

サーバプログラム間通信の処理オーバヘッドの一つとして，OS サーバ間での授受データの複写がある．この処理オーバヘッドへの対処として，データ複写レスによる通信がある．これは，仮想アドレスと実アドレスのアドレス変換表（以降，ページテーブル）の書き換えにより，仮想空間の当該領域を 2 仮想空間間で貼り替え，または共有することにより実現されている [5]−[7]．しかし，組込みシステムのプロセッサでは，MMU の機能は低機能であることが多く，データ複写レスであっても通信時に発生する TLB（Translation Lookaside Buffer）ミスに伴う処理オーバヘッドが大きい．例えば，Pentium4 プロセッサでは，TLB ミスが発生した場合，ハードウェアがページテーブルを探索し，ページテーブルの当該のエントリ情報を TLB へ登録する．一方，代表的な組込みプロセッサの一つである SH-4 では，TLB ミス発生時に例外が発生し，ソフトウェアへ処理が移行する．このため，Pentium4 と比較し，SH-4 では，TLB ミスの発生に伴う処理のオーバヘッドが大きい．

そこで，マイクロカーネル OS における低機能 MMU 制御法を提案する．具体的には，TLB の扱いを工夫する．提案制御法は，サーバプログラム間通信で利用する領域については，ページテーブルのエントリを利用せず，TLB エントリでページの割り当てを管理する．これにより，サーバプログラム間通信で利用する領域において TLB ミスは発生しないため，サーバプログラム間通信を高速化できる．SH-4 を例として，提案制御法をマイクロカーネル構造を有する ***AnT*** オペレーティングシステム（An operating system with Adaptability and Toughness）（以降，***AnT*** と略す）[7] に実現する方式を示し，サーバプログラム間通

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

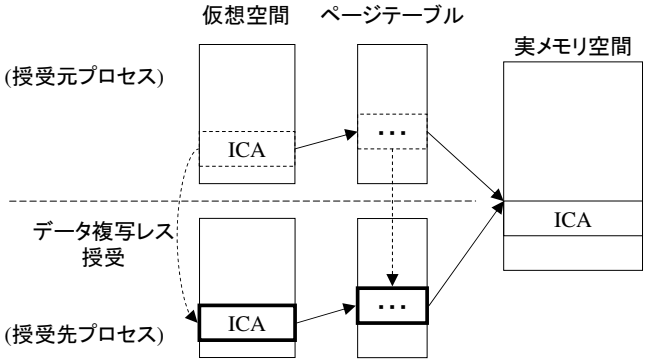


図 1 データ複写レス授受の様子

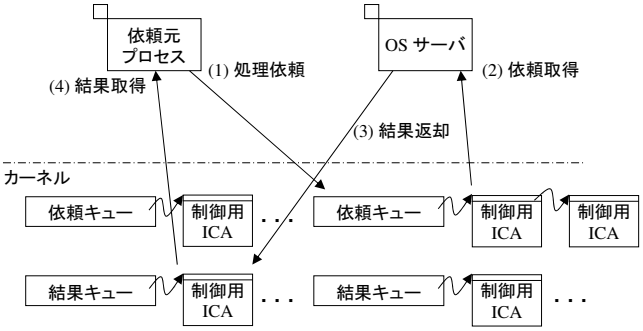


図 2 サーバプログラム間通信の基本機構

信性能を評価した結果を報告する．さらに，パケット送信処理性能について，***AnT*** と既存のモノリシックカーネル OS の ART-Linux を比較した結果を報告する．

2. *AnT* オペレーティングシステム

2.1 データ複写レス授受機能

AnT はマイクロカーネル構造を有するオペレーティングシステムである．プロセス間の通信を高速化するため，コア間通信データ域（ICA:Inter-core Communication Area）を利用したデータ複写レス授受機能を有する．ICA の特徴として以下の 3 つがある．

(1) ページを単位とし，n ページ分の領域の確保と解放
(2) 確保した領域 (n ページ) の実メモリ連続の保証
(3) 2 仮想空間での領域の貼り替え

ICA は，ページを最小単位として管理される領域であり，ICA へのアクセスは，プロセスごとの仮想空間のページテーブルを通して行われる．ここで，ページテーブルへの書き込みを貼り付けと呼び，ページテーブルからの削除を剥がしと呼ぶ．また，貼り替えとは，剥がしと貼り付けを行うことを意味する．プロセス間の複写レスでのデータ授受の様子を**図 1** に示す．ICA を利用したプロセス間でのデータ授受は，授受するデータを格納した ICA をデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間へ貼り付けることで行われる．

2.2 サーバプログラム間通信機構 [7]

サーバプログラム間通信の基本機構を**図 2** に示す．ICA を利用することにより，プロセス間でデータ複写レスでの通信を実現している．具体的には，OS サーバへ渡す引数や通信制御の情報（以降，依頼情報）を制御用の ICA（以降，制御用 ICA）に格納し，扱うデータをデータ用の ICA（以降，データ用 ICA）に格納する．カーネルは，各プロセスごとに通信のための依頼キューと結果キューを持つ．基本的な通信の流れを以下に述べる．

(1) 依頼元プロセスが処理依頼を行うと，カーネルは OS サーバの依頼キューに依頼情報を格納した制御用 ICA を登録し，OS サーバへ制御用 ICA を貼り替える．
(2) OS サーバは，依頼キューから依頼情報を格納した制御用 ICA を取得し処理を実行する．
(3) OS サーバが結果返却を行うと，カーネルは依頼元プロセスの結果キューに結果情報を格納した制御用 ICA を登録し，依頼元プロセスへ制御用 ICA を貼り替える．
(4) 依頼元プロセスは，結果キューから結果情報を格納した制御用 ICA を取得し処理を終了する．また，同期型と非同期型の通信インタフェースを同様な形式で提供し，両インタフェースを選択して利用できる．

マイクロカーネル OS では，OS サーバを多段に経由して処理依頼が発行される．これを多段依頼と名付ける．このとき，制御用 ICA を処理依頼のたびに確保すると処理オーバヘッドが大きい．そこで，1 つの制御用 ICA を持ち回り，依頼情報を積み重ねることでオーバヘッドを抑制する．また，多段依頼された処理は，積み重ねられた依頼情報を基に中継した OS サーバを経由した逐次的な返却が行われる．しかし，必ずしも逐次的な返却を行う必要がない場合は，依頼元のプロセスへ直接返却することにより，処理を高速化する．具体的には，制御用 ICA に flag を設け，返却の要否を設定する．結果返却時にカーネルが flag を確認し，返却が必要な依頼元のプロセスに直接返却を行う．

3. 低機能 MMU 制御法

3.1 考え方

データ複写レス通信法の問題点は大きく二つある．一つは，ページテーブルの書き換え処理を行う必要があることである．もう一つは，TLB ミス時の処理が必要であり，高機能 MMU ではハードウェアが行うが，低機能 MMU ではソフトウェアが行う必要がありオーバヘッドが大きいことである．さらに，共有方式ではプロセス間でのデータ保護ができない問題がある．

これらに対し，低機能 MMU の特徴である「TLB への情報登録をソフトウェアで行える」ことを生かして制御を行う．具体的には，サーバプログラム間通信でデータ授受に利用する領域（以降，データ通信域と呼ぶ）へのアクセスに対し，TLB ミスが発生しないように，TLB への情報

登録を工夫する．これにより，ページテーブルの書き換え処理を不要とし，TLB ミス発生はデータ保護例外として扱えるようにする．

これにより，以下が期待できる．

(1) ページテーブルの書き換え処理が不要である．

(2) TLB ミスは基本的に発生しないため，処理オーバヘッドが大きく減少する．

(3) TLB ミス発生はデータ保護例外であり，プロセス間でのデータ保護ができる．

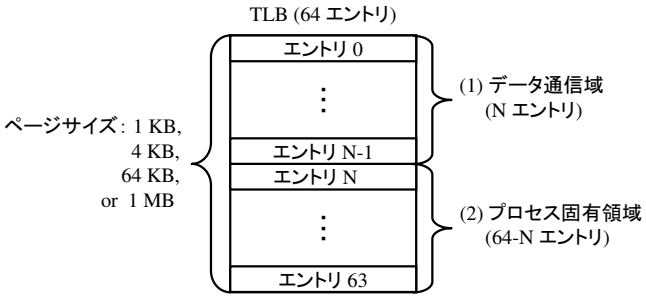


図 3 TLB エントリの分割管理

3.2 制御法

提案制御法では，データ通信域については，ページテーブルを利用せず，TLB エントリでページ割り当てを管理する．つまり，データ通信域を貼り付ける場合には，TLB エントリを確保し，ページテーブルのエントリ情報に相当する情報（以降，アドレス変換情報と呼ぶ）を TLB エントリへ登録する．また，データ通信域を剥がす場合には，アドレス変換情報を TLB エントリから削除し，TLB エントリを解放する．これにより，ページテーブルの書き換えが不要となる．また，データ通信域に対する TLB ミスは発生しなくなる．さらに，プロセスが自プロセスへ貼り付いていないデータ通信域へアクセスした場合，TLB ミスが発生する．このため，データ通信域に対する TLB ミスをデータ保護例外として扱うことにより，データ通信域への不正なアクセスやバグなどによるメモリ破壊を防止できる．

以降の記述では，代表的な組込みプロセッサの一つである SH-4 を例として，その MMU について，提案制御法を具体的に記述する．SH-4 の MMU の特徴を以下に示す．

（特徴 1）多重仮想記憶を利用する場合，仮想空間は 5 つの領域に分かれており，各領域でアクセス権限やアドレス変換方式が異なるため，使用方法が限定される．例えば，カーネル空間として利用される領域は，実メモリにストレートマッピングされ，ページテーブルを通さずにアクセスされる．

（特徴 2）一つの仮想空間内において，ページサイズを 1KB，4KB，64KB，および 1MB から選択でき，各ページサイズを混在して利用できる．

（特徴 3）TLB は 64 エントリで構成され，ページテーブルのエントリ情報を TLB へ登録する際，どのエントリを利用するか指定できる．なお，指定しない場合，利用するエントリは，ハードウェアのランダムカウンタの値により決定される．

（特徴 4）多重仮想記憶を利用する場合，TLB のエントリ情報として各仮想空間の識別子を保持する．このため，仮想空間切り換えの際に必ずしも TLB をフラッシュする必要がない．

これらの特徴を生かした制御を以下に述べる．TLB エントリをデータ通信域と，プロセスのテキスト部やデータ部などの領域（以降，プロセス固有領域と呼ぶ）に利用する部分に分割し，管理する．TLB エントリの分割管理の様子を**図 3** に示し，以下で説明する．

(1) 0 から $N-1$ エントリをデータ通信域へ割り当てる．この結果，利用できるデータ通信域のページ数は，最大 N 枚になる．

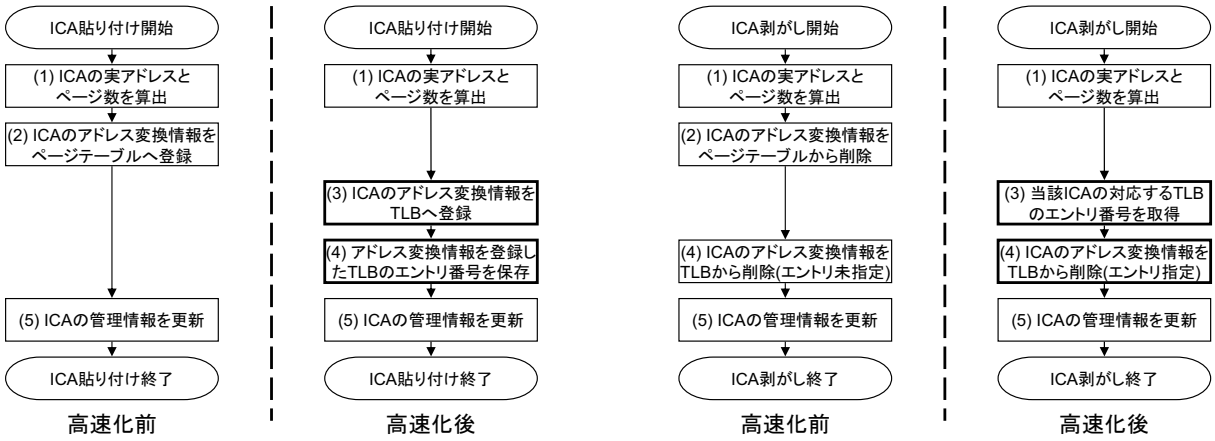
(2) 残りの N から 63 エントリをプロセス固有領域に割り当てる．

なお，プロセス固有領域に割り当てた TLB エントリの利用状態を管理し，TLB ミスを抑制する．具体的には，TLB へのアドレス変換情報の登録を行う際，利用されていないエントリを指定する．全エントリが利用されている場合は，ハードウェアのランダムカウンタを用いてエントリを決定する．これにより，全エントリをランダムに利用するよりも TLB ミスの発生を抑制できる．さらに，データ通信域へ割り当てる TLB エントリの個数 N，およびページサイズは，サーバプログラム間通信機構の利用形態を考慮し，動作環境にあわせて決定する．これにより，動作環境にあわせ，最大限 TLB ミスを発生させないような制御ができる．

3.3 *AnT* への実現

AnT は，データ通信域として，ICA を有する．このため，ICA 貼り付けと ICA 剥がしの処理を低機能 MMU 制御法に基づき実現する．以降，低機能 MMU 制御法を実現していない ***AnT*** を高速化前の ***AnT***，実現した ***AnT*** を高速化後の ***AnT*** と呼ぶ．なお，高速化前の ***AnT*** は，SH-4 上で動作する ART-Linux と同様にハードウェアのランダムカウンタで利用する TLB エントリを決定し，ハードウェアの TLB の全エントリを探索し削除する機能で TLB エントリのアドレス変換情報の削除を行う．また，3.2 節の（特徴 4）を利用し，プロセス切換における TLB フラッシュを不要としている．

ICA 貼り付けと ICA 剥がしの処理流れを**図 4** に示す．高速化後は，ページテーブルを用いずに ICA のページ割り当てを管理するため，ICA のアドレス変換情報をページテーブルへ登録，および削除する処理（2）を行わない．また，ICA 貼り付けにおいて，ICA のアドレス変換情報を



（A）ICA 貼り付けの処理流れ　　（B）ICA 剥がしの処理流れ

図 4　ICA 貼り付けと ICA 剥がしの処理流れ

表 1　ファイル数とコード量

	ファイル数（個）	コード量（行）
高速化前	129	18175
高速化後	129	18265
追加・変更部分	5 (3.8%)	205(1.1%)

TLB へ登録する処理（3），およびアドレス変換情報を登録した TLB のエントリ番号を保存する処理（4）を行う．さらに，ICA 剥がしにおいて，当該 ICA の対応する TLB のエントリ番号を取得する処理（3）を行う．なお，ICA 剥がしにおける ICA のアドレス変換情報を TLB から削除する処理（4）について，高速化前は，ハードウェアの TLB の全エントリを探索し削除する機能を利用し，当該のエントリ情報を削除する．一方，高速化後は，当該のエントリを直接指定し，エントリ情報を削除する．

4. 評価

4.1 工数

提案制御法の実現に要した工数を明らかにする．具体的には，高速化により追加・変更したファイル数とコード量について評価する．高速化前の ***AnT***，高速化後の ***AnT***，および追加・変更した部分のファイル数とコード量を**表 1** に示す．表 1 より，高速化するために追加・変更したファイル数とコード量は，それぞれ 5 個と 205 行であり，高速化前のファイル数とコード量のそれぞれ約 3.8%と約 1.1%にあたる．これより，MMU の機能に依存する部分を局所化できているといえる．

4.2 性能

4.2.1 環境

評価環境を**表 2** に示す．***AnT*** との比較対象として ART-Linux を利用する．なお，***AnT*** と ART-Linux は，同一のコンパイラ（gcc-3.2.3）を用いてコンパイルした．また，***AnT*** は，32 個の TLB エントリを ICA へ割り当て，残りの 32 個をプロセス固有領域に割り当て，ページサイズを 4KB 固定とした．なお，測定結果は，10 回試行した場合の平均処理時間である．

表 2　評価環境

	評価（送信）計算機	受信計算機
OS	***AnT***	FreeBSD 4.3-R
	or ART-Linux（2.4.29）	
CPU	SH-4（SH7751R）	Pentium4
Frequency	240 MHz	3.4 GHz
Memory	32 MB	1024 MB
NIC	Intel GD82551ER	Intel 82558

4.2.2 予備評価

AnT のサーバプログラム間通信機構の処理オーバヘッドについて考える．***AnT*** のサーバプログラム間通信機構は，同期の処理依頼と結果返却，および非同期の処理依頼と結果返却という 4 つの基本処理からなる．また，それぞれの処理について，データ用 ICA の授受を行う場合と行わない場合がある．さらに，基本処理は，以下の部分処理からなる．

(1) ICA 貼り付け
(2) ICA 剥がし
(3) システムコール発行
(4) ICA への初アクセス
(5) プロセス切換

ここで，システムコール発行は，プロセスからカーネルへの処理移行とカーネルからプロセスへの処理の戻りを合わせた処理である．また，ICA への初アクセスは，そのプロセスに ICA が貼り付けられた後，初めてアクセスした際に発生する処理である．

高速化前の ***AnT*** の基本処理の処理オーバヘッドを分析する．基本処理の測定の様子を**図 5** に示す．基本処理の測定は，依頼元プロセスと OS サーバで通信を行い，各処理時間を測定する．なお，測定において，処理依頼は，OS サーバへ渡す引数および戻り値を無しとし，データ用 ICA のサイズを 4KB にした．また，通信機構のオーバヘッド

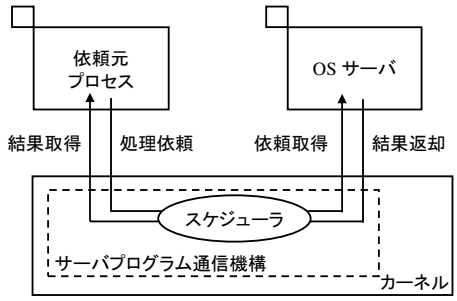


図 5 基本処理の測定の様子

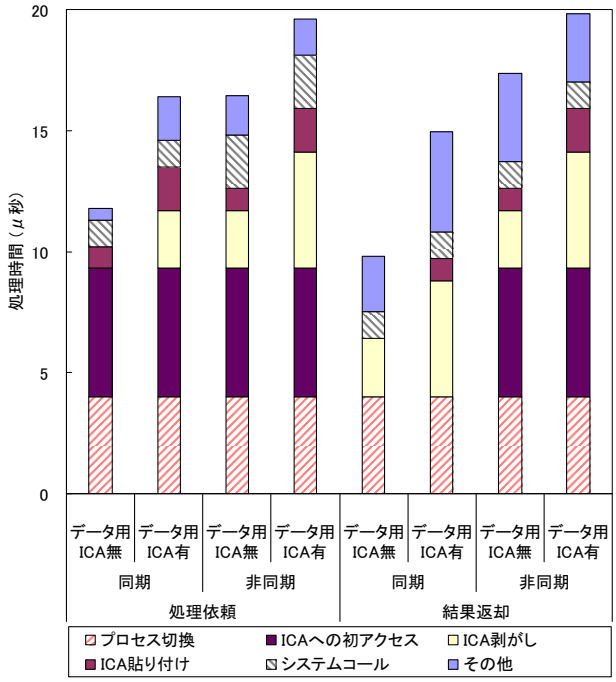


図 6 高速化前の基本処理の処理時間

を明確化するため，OS サーバでは通信に関連する処理以外の処理は行わない．

高速化前の基本処理の処理時間を**図 6** に示す．図 6 より，各基本処理の処理時間の約 23〜58%は，ICA 貼り付け，ICA 剥がし，および ICA への初アクセスの処理時間である．ここで，ICA 貼り付けと ICA 剥がしは，通信中に頻発する．このため，ICA 貼り付けと ICA 剥がしの処理オーバヘッドを削減することにより，高速化が期待できる．また，ICA への初アクセスは，TLB ミスによるオーバヘッドであるため，TLB ミスの発生を抑制することにより，高速化が期待できる．

4.2.3 基本性能

高速化後の性能を明らかにするため，***AnT*** のサーバプログラム間通信の部分処理の処理時間，基本処理の処理時間，および特徴的な機能の直接返却について評価する．まず，各部分処理の処理時間について述べる．部分処理の処理時間を**表 3** に示す．表 3 より，以下のことがわかる．

(1) 高速化後の ICA 貼り付けの処理時間は，高速化前から 0.1 μ秒（約 11%）増加している．これは，高速化後では，

表 3 部分処理の処理時間

処理内容	高速化前	高速化後
ICA 貼り付け	0.9 μ秒	1.0 μ秒
ICA 剥がし	2.4 μ秒	1.0 μ秒
システムコール発行	1.1 μ秒	1.1 μ秒
ICA への初アクセス	5.3 μ秒	0.5 μ秒
プロセス切換	4.0 μ秒	4.0 μ秒

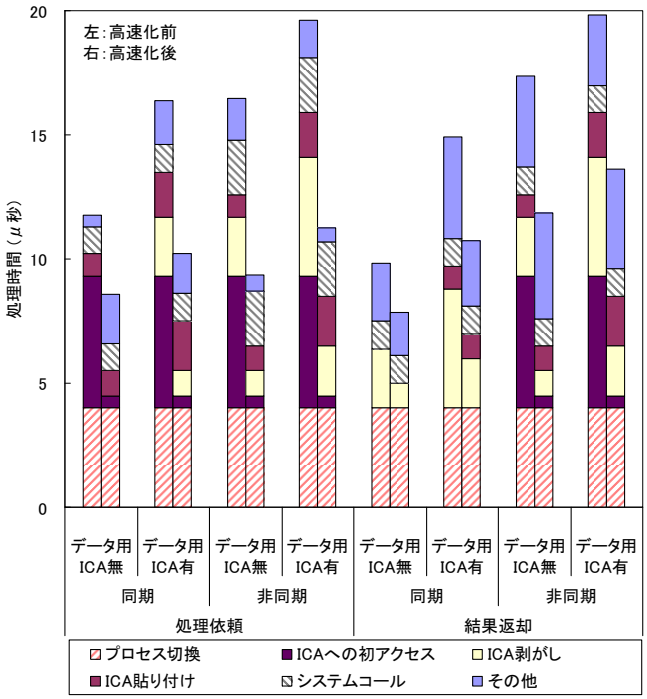


図 7 高速化後の基本処理の処理時間

TLB へ ICA のアドレス変換情報を登録する処理を追加したためである．しかし，高速化後では，ページテーブルの書き換え処理を行わないため，処理時間の増加量は小さい．

(2) 高速化後の ICA 剥がしの処理時間は，高速化前に比べ，1.4 μ秒（約 58%）減少している．これは，高速化前と高速化後では，ICA 剥がし処理において，TLB のエントリ情報を削除する処理が異なるためである．高速化前は，ハードウェアの持つ TLB の全エントリを探索し削除する機能を利用し，当該のエントリ情報を削除する．一方，高速化後は，ソフトウェアで TLB のエントリ情報を管理しているため，当該のエントリを指定し，エントリ情報を削除する．

(3) 高速化後の ICA への初アクセスの処理時間は，高速化前に比べ，4.8 μ秒（約 91%）減少している．これは，高速化後では，ICA への初アクセスにおいて，TLB ミスが発生しないためである．

次に，基本処理の処理時間について述べる．基本処理の測定方法は，4.2.2 項と同様の方法で行った．基本処理の処理時間を**図 7** に示す．図 7 には，高速化の効果を明らかにするため，高速化前の処理時間も記載している．図 7 より，高速化により，処理依頼時間は約 3〜8 μ秒（約 27〜

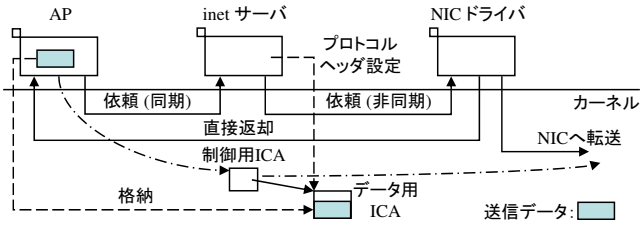


図 8 ***AnT*** における TCP/IP 通信

43%），結果返却時間は約 2〜6 μ秒（約 20〜32%）減少した．これは，ICA 剥がし，および ICA への初アクセスの処理時間が減少したためである．なお，図 7 のデータ用 ICA 有の場合，データ用 ICA へのアクセスは発生していない．これは，測定時に OS サーバは通信に関連する処理以外の処理を行なっていないためである．一方，実サービスにおいては，データ用 ICA へのアクセスが発生する．このとき，データ用 ICA への初アクセスの際，高速化前は TLB ミスが発生するのに対し，高速化後は発生しない．

最後に，直接返却について述べる．直接返却を利用した場合の処理時間について，OS サーバを n（n = 1〜5）段介した処理依頼において，AP プロセスが処理依頼を発行してから結果が返ってくるまでの処理時間を評価した．測定の結果，高速化により段数に関係なく，逐次返却時間，ならびに直接返却時間を約 17〜32%短縮できることを確認した．

4.2.4 TCP/IP 通信性能

AnT における TCP/IP 通信によるパケット送信処理の流れを**図 8** に示す．***AnT*** は，ネットワークプロトコル制御サーバと NIC（Network Interface Controller）ドライバプロセスの 2 つのプロセスにより TCP/IP 通信機能を提供している．このため，***AnT*** のパケット送信処理では，モノリシックカーネル OS に比較し，サーバプログラム間通信の処理オーバヘッドが発生する．しかし，***AnT*** では，あらかじめデータ用 ICA へ送信データを格納し，これをもとにパケット送信処理を呼び出すことにより，パケット生成処理とデータ複写によるオーバヘッドを削減できる．

TCP/IP 通信におけるパケット送信処理時間について評価する．また，ART-Linux と比較評価する．高速化前の ***AnT***，高速化後の ***AnT***，および ART-Linux のパケット送信処理時間を**図 9** に示す．なお，評価では，評価（送信）計算機と受信計算機をハブを介して接続し，1 パケットの送信処理時間を測定した．また，実環境を想定し，共存プロセスを一つ走行させた場合のパケット送信処理時間を**図 10** に示す．なお，共存プロセスは， 32KB（SH7751R のデータキャッシュと同サイズ）の領域に対し，メモリアクセス（Read/Write）を繰り返すものである．なお，共存プロセスの優先度は，全プロセス中最低とした．また，送信するデータサイズは，256B，512B，768B，1024B，および 1280B とした．図 9，図 10 より，以下のことがわかる．

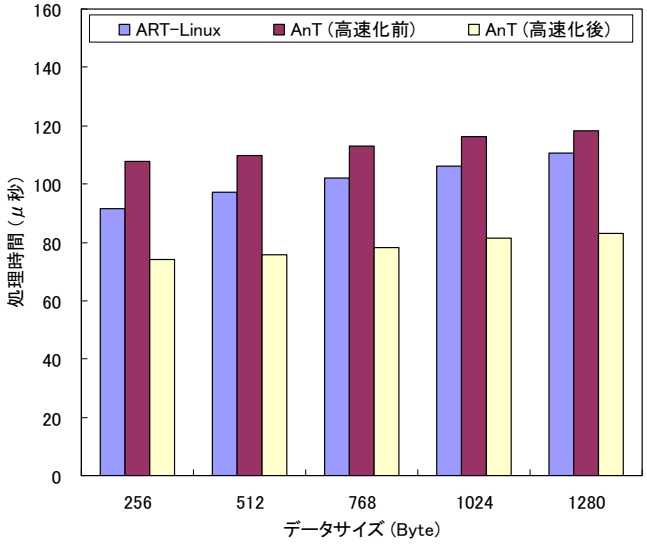


図 9 パケット送信処理時間

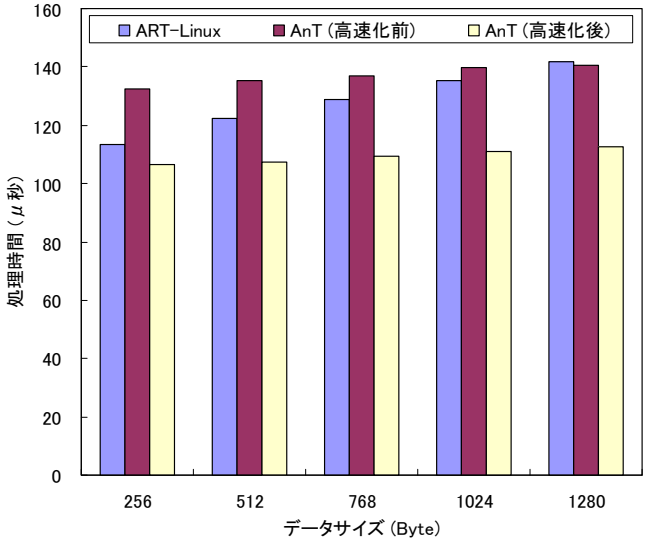


図 10 共存プロセスを走行させた場合のパケット送信処理時間

(1) 高速化後の ***AnT*** のパケット送信処理時間は，高速化前のパケット送信処理時間に比べ，約 33〜36 μ秒（約 30〜31%）短い．これは，サーバプログラム間通信の高速化により，サーバプログラム間通信の処理オーバヘッドが減少したためである．また，データ用 ICA への初アクセスに伴う処理オーバヘッドの差によるものである．高速化前は，データ用 ICA への初アクセスに伴い，TLB ミスが発生するのに対し，高速化後は発生しない．

(2) 高速化後の ***AnT*** におけるパケット送信処理時間は，ART-Linux のパケット送信処理時間に比べ，約 17〜28 μ秒（約 19〜25%）短い．これは，***AnT*** は，パケット送信処理中に，パケット生成処理を行なわないためである．ART-Linux 等のモノリシックカーネル OS では，パケット送信処理において，ユーザ空間からカーネル空間へ送信データを複写し，パケットを生成する．一方，***AnT*** では，

あらかじめデータ用 ICA へ送信データを格納し，これをもとにパケット送信処理を呼び出す．

(3) 高速化後の ***AnT*** は，ART-Linux に比べ，送信データのサイズの増加に伴うパケット送信処理時間の増加量は小さい．これは，送信データの複写回数の違いによるものである．***AnT*** は，データ複写レス通信により，ART-Linux に比べ，送信データの複写回数が少ない．

(4) 共存プロセスを走行させた場合，パケット送信処理時間は増加する．これは，共存プロセスの走行に伴い，TLB ミス等のキャッシュミスが増加したためだと考えられる．

(5) ART-Linux において共存プロセスを走行させた場合，共存プロセスを走行させない場合に比べ，送信データサイズの増加に伴うパケット送信処理時間の増加量は大きくなる．これは，共存プロセスの走行により，送信データの複写処理において TLB ミス等のキャッシュミスが頻発したためだと考えられる．一方，***AnT*** では，共存プロセスの有無にかかわらず，送信データサイズの増加に伴うパケット送信処理時間の増加量は，一定である．これは，複写レスデータ授受により，データ複写回数が ART-Linux よりも少なく，共存プロセスの走行による影響が小さいためだと考えられる．このため，共存プロセスを走行させた場合，1280B のデータ送信処理は，ART-Linux よりも高速化前の ***AnT*** が高速である．

以上のことから，MMU の特徴を生かすことにより，マイクロカーネル OS において，既存のモノリシックカーネル OS よりも高速なパケット送信処理を実現できる．

5. おわりに

低機能 MMU 制御法を提案し，提案制御法の ***AnT*** への実現と評価結果を述べた．提案制御法は，低機能 MMU の特徴である「TLB への情報登録をソフトウェアで行える」ことを生かし，データ通信域についてはページテーブルを利用せず，TLB エントリでページ割り当てを管理する．これにより，データ通信域への不正なアクセスやバグなどによるメモリ破壊を防ぎ，信頼性を低下させず，サーバプログラム間通信の高速化を実現している．

評価では，工数，サーバプログラム間通信性能，およびパケット送信処理性能について評価した．高速化するために追加・変更したファイル数とコード量は，高速化前のファイル数とコード量のそれぞれ約 3.8%と約 1.1%にあたることを確認した．また，サーバプログラム間通信の処理依頼時間で平均約 38%，結果返却時間で平均約 28%短縮できることを示した．さらに，OS サーバを n 段介した処理依頼において，段数に関係なく，逐次返却と直接返却の時間を約 17〜32%短縮できることを示した．最後に，マイクロカーネル OS において，既存のモノリシックカーネル OS よりも高速なパケット送信処理を実現できることを明らかにした．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（課題番号：24300008）による．

参考文献

[1] Hildebrand, D.: An Architectural Overview of QNX, *Proc. Workshop on Micro-kernels and OtherKernel Architectures*, pp.113-126 (1992).
[2] Liedtke, J.: Toward Real Microkernels, *Comm. ACM*, Vol.39, No.9, pp.70-77 (1996).
[3] Tanenbaum, A.S., Herder, J.N. and Bos, H.: Can we make operating systems reliable and secure?, *IEEE Computer Magazine*, Vol.39, No.5, pp.44-51 (2006).
[4] 山田晋平，中本幸一：組込みシステム向け複数メモリ領域間保護機能，電子情報通信学会論文誌 D，Vol.J93-D，No.10，pp.2011-2020 (2010).
[5] Black, D.L., Golub, D.B., Julin, D.P., Rashid, R.F., Draves, R.P., Dean, R.W., Forin, A., Barrera, J., Tokuda, H., Malan, G. and Bohman, D.: Microkernel Operating System Architecture and Mach, *J. Inf. Process.*, Vol.14, No.4, pp.442-453 (1992).
[6] 毛利公一，大久保英嗣：マイクロカーネル Lavender の設計と開発，電子情報通信学会論文誌 D-I，Vol.J82-D-I，No.6，pp.730-739 (1999).
[7] 岡本幸大，谷口秀夫：***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価，電子情報通信学会論文誌 D，Vol.J93-D，No.10，pp.1977-1989 (2010).