

AnT オペレーティングシステムにおける サーバプログラム間通信機構の評価

岡本 幸大[†] 谷口 秀夫^{††}

^{††} 岡山大学大学院自然科学研究科

〒700-8530 岡山県岡山市津島中三丁目一番一号

E-mail: tokamoto@swlab.cs.okayama-u.ac.jp, tani@cs.okayama-u.co.jp

あらまし 組み込みシステムでは、提供するサービスや利用できる資源は限られており、OSに必要な機能も限られる。そこで、システムに基づいて必要な機能のみを有するOSを利用することで、メモリをはじめとする資源を有効に活用できる。マイクロカーネル構造はOSの機能をシステムが提供するサービスに適應させることが可能なプログラム構造である。しかしながら、マイクロカーネル構造はサーバ間の通信が頻発し性能が低下してしまう。そこで、我々は性能低下を抑えるための高速なプログラム間通信機構を*AnT*オペレーティングシステム上で設計した。*AnT*はマイクロカーネル構造を有する。ここでは、サーバプログラム間通信の基本処理評価を行い、そのオーバヘッドを明確にする。また、サービス評価として、本機構上にネットワーク通信機能を行うためのプロトコルスタック、およびNICドライバを実装し、そのスループットを測定する。

キーワード *AnT* オペレーティングシステム, マイクロカーネル, プロセス間通信

Evaluation of Communication Mechanism between Server Programs for *AnT* Operating System

Kouta OKAMOTO[†] and Hideo TANIGUCHI^{††}

^{††} Graduate School of Natural Science and Technology Okayama University

1-1, Tsushima-Naka, 3-Chome, Okayama 700-8530 Japan

E-mail: tokamoto@swlab.cs.okayama-u.ac.jp, tani@cs.okayama-u.co.jp

Abstract Embedded systems usually have limited memory resources. On the resource-constrained embedded systems, OS kernels should be flexibly configurable to fit any field. The microkernel architecture is suitable for the embedded systems because of a small footprint core and flexibly selective server programs. However, microkernel systems tend to have performance problems on inter-server communication. To solve the performance issue, we designed a high-performance inter-process communication framework and implemented that on the *AnT* microkernel. In this paper, we show a basic performance evaluation and a practical in-system throughput of our implementation.

Key words *AnT* operating system, microkernel, inter-process communication

1. ま え が き

近年、計算機の高機能化に伴い、基盤ソフトウェアであるオペレーティングシステム(OS)の機能も高度化し複雑化している。これは、計算機が提供する多くのサービスに対応するために、OSもまた多くの機能を有することに起因する。一方、組み込みシステムを考えた場合、提供するサービスや利用できる資源は限られており、OSに必要な機能も限られる。そこで、システムが提供するサービスに合わせ、必要な機能のみを有する

OSを利用することにより、メモリをはじめとする資源を有効に活用できる。

OSの機能をシステムが提供するサービスに適應させるプログラム構造として、マイクロカーネル構造[1][2]がある。マイクロカーネル構造は、割り込み処理や例外処理といった最小限のOS処理をカーネル(マイクロカーネル)として実現し、ファイル管理処理や通信処理などの処理をプロセスとして実現するプログラム構造である。つまり、多くのOS機能は、カーネル外にプロセスとして実現する。以降、これをOSサーバと名付け

表 1 インタフェース形式の機能と形式

機能	インタフェース
サーバ登録	registserver(coreid); coreid; 登録するサーバのコア ID
同期処理依頼	callsync(p); p; 依頼情報を格納した制御用 ICA のアドレス
非同期処理依頼	callsync(p); p; 依頼情報を格納した制御用 ICA のアドレス
結果返却	ret(p); p; 返却情報を格納した制御用 ICA のアドレス
要求取得 & 結果返却	get(p); p; 返却情報を格納した制御用 ICA のアドレス または NULL

制御用 ICA に格納し、格納した制御用 ICA を引数に callsync() システムコールを発行する。これにより、サーバプロセスの依頼キューへ依頼が登録される。依頼元プロセスはサーバプロセスの処理が終了するまで待ち状態へ移行する。

callsync() は、サーバプロセスに対し非同期的に処理依頼を発行する操作である。処理は callsync() と同様であるが、依頼元プロセスへはサーバプロセスの処理終了を待たずに制御が戻る。

ret() は、結果返却を行なう操作である。サーバプロセスは依頼された処理を終了すると、結果情報を格納した制御用 ICA を引数に指定し、ret() を発行する。これにより、依頼元プロセスへ結果が返却される。

get(), は結果返却、および依頼または結果を取得する操作である。サーバプロセスは引数に NULL を指定して get() システムコールを発行することで、依頼または結果を取得する。依頼または結果が登録されていない場合、登録されるまで待ち状態へ移行する。引数に返却情報を格納した制御用 ICA を指定した場合、結果返却を行なった後、依頼または結果の取得を行なう。

なお、ret() は複数の要求を保持するサーバで使われ、通常のサーバでは get() を使用する。

3. 基本性能

3.1 項目と環境

サーバプログラム間通信制御の基本的な処理として、同期処理依頼とその返却、および非同期処理依頼とその返却がある。また、それぞれの処理においてデータ用 ICA の授受を行なう場合と行なわない場合が存在する。これらについて性能を明らかにする。

具体的には、AnT オペレーティングシステムを Pentium4(2.8GHz) プロセッサの計算機上で走行させ、各処理時間を測定した。測定には、図 3 に示すように、2つのプロセス間の通信処理を用いた。なお、測定には RDTSC 命令を使用し、測定結果は 10 回試行した場合の平均処理時間である。

3.2 同期処理依頼

同期処理依頼における処理依頼と結果返却の処理時間を評価する。同期処理依頼では、制御用 ICA は依頼元プロセスへ貼り付けたままとし、依頼取得時にサーバプロセスへ貼り付ける。

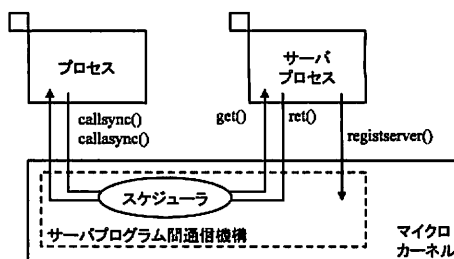


図 3 測定環境

これにより制御用 ICA 貼り替えのオーバーヘッドを削減し、効率化を図っている。また、依頼元プロセスは制御用 ICA を保持したまま返却待ち状態となっているため、結果返却時にはサーバプロセスから制御用 ICA を剥がした後、待ち状態の依頼元プロセスを起こすだけでよい。これにより結果キューへの結果登録、および結果キューからの結果取得によるオーバーヘッドを削減している。

同期処理依頼と結果返却の処理流れを図 4 と図 5 に示す。同期処理依頼として、図 4 における a1-a2 間の処理時間を測定した。具体的には、依頼元プロセスが callsync() を発行してから、サーバプロセスが get() により処理依頼を取得するまでの処理時間を測定した。サーバプロセスはあらかじめ get() を発行し処理取得待ち状態となっており、その後依頼元プロセスが callsync() を発行するものとする。また、同期処理依頼における結果返却として、図 5 における b1-b2 間の処理時間を測定した。具体的には、サーバプロセスが同期的に依頼された処理を終了した後、結果を依頼元へ返却するために get() を発行してから依頼元に結果が返るまでである。依頼元プロセスは callsync() を発行した場合、サーバ処理の終了待ち状態となっている。このため、結果返却は待ち状態の依頼元プロセスを起こすことで実現される。

測定結果を表 2 に示す。表 2 より、以下のことが分かる。

(1) 処理依頼（データ用 ICA なし）における処理時間は 15.47 μ 秒、結果返却（データ用 ICA なし）における処理時間は 11.96 μ 秒である。また、データ用 ICA の有無による差分から、データ用 ICA の授受にかかるオーバーヘッドは約 0.7~0.8 μ 秒である。データ用 ICA の授受オーバーヘッドは、各処理時間の 4~6% であり、ICA によるデータ転送は効率的であると言える。

また、各処理の分析により、キュー操作前後の割り込みマスク処理 1 回につき約 1.57 μ 秒のオーバーヘッドが発生しており、割り込みマスク処理は処理依頼で 8 回、結果返却で 6 回呼ばれていた。このことから、以下のことが分かる。

(2) 割り込みマスク処理により、処理依頼では $8 \times 1.57 = 12.56$ μ 秒、結果返却では $6 \times 1.57 = 9.42$ μ 秒のオーバーヘッドが発生している。割り込みマスク処理のオーバーヘッドは各処理時間の約 80% であり、割り込みマスク処理が与える影響が大きいがわかる。割り込みマスク処理では割り込みコントローラに対して I/O 処理を行っており、これがオーバーヘッドの要因であると推察する。

表 3 非同期処理依頼における測定結果 (単位: μ 秒)

	データ用 ICA なし	データ用 ICA あり	差分
処理依頼	19.7	20.4	0.7
結果返却	18.89	19.73	0.84

要がある。さらに、依頼元プロセスは、get() を発行し返却された結果を結果キューから取得しなければならない。このため、同期処理依頼よりも非同期処理依頼は処理時間が增大すると予想できる。

非同期処理依頼と結果返却の処理流れを図 6 と図 7 に示す。非同期処理依頼として、図 6 における a1'-a2' 間の処理時間を測定した。具体的には、依頼元プロセスが callasync() を発行してから、サーバプロセスが get() により処理依頼を取得するまでである。同期処理依頼と同様に、サーバプロセスはあらかじめ get() を発行し処理取得待ち状態となっており、その後依頼元プロセスが callasync() を発行するものとする。また、非同期処理依頼における結果返却として、図 7 における b1'-b2' 間の処理時間を測定した。具体的には、サーバプロセスが非同期的に依頼された処理を終了し、結果を依頼元へ返却するために get() を発行してから依頼元に結果が返るまでである。依頼元プロセスは非同期処理依頼を発行した場合、サーバ処理の終了を待たずに元の処理へ復帰する。このため、結果返却は依頼元プロセスが get() を発行し結果を取得することで実現される。

測定結果を表 3 に示す。表 3 より、以下のことが分かる。

(1) 処理依頼 (データ用 ICA なし) における処理時間は 19.7 μ 秒、結果返却 (データ用 ICA なし) における処理時間は 18.89 μ 秒である。また、データ用 ICA の有無による差分から、データ用 ICA の授受にかかるオーバーヘッドは約 0.7~0.8 μ 秒である。データ用 ICA の授受オーバーヘッドは、各処理時間の 3~4% であり、ICA によるデータ転送は効率的であるとと言える。

また、同期処理依頼と同様に、割り込みマスク処理オーバーヘッドを調査したところ、割り込みマスク処理は処理依頼と結果返却で各 10 回呼ばれていた。このことから、以下のことが分かる。

(2) 割り込みマスク処理により、処理依頼と結果返却で $10 \times 1.57 \mu$ 秒 = 15.7 μ 秒のオーバーヘッドが各々発生している。割り込みマスク処理のオーバーヘッドは各処理時間の約 80% であり、同期処理依頼と同様に割り込みマスク処理が与える影響が大きいことがわかる。

3.4 考 察

同期処理依頼と非同期処理依頼の処理時間を比較した場合、処理依頼と結果返却ともに非同期処理依頼の方が処理オーバーヘッドが大きく、各処理時間の差分は処理依頼で 4.23 μ 秒、結果返却で 6.93 μ 秒であった。処理依頼と結果返却の差分に差が見られる理由として、割り込みマスク処理の呼び出し回数の違いが考えられる。割り込みマスク処理の呼び出し回数の差は、処理依頼では 2 回であるのに対し、結果返却では 4 回である。このため、割り込みマスク処理 2 回分の差が生じたといえる。

また、割り込みマスク処理を除いた場合の処理時間の差分は

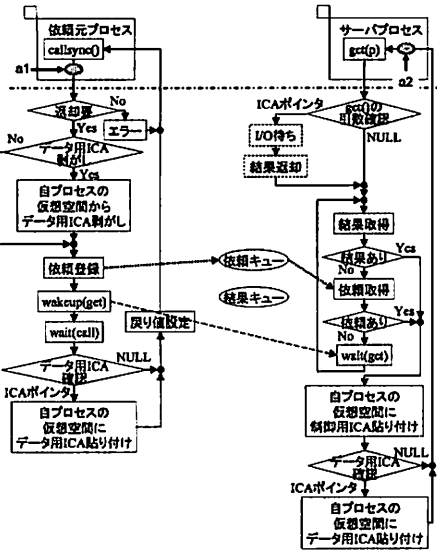


図 4 同期処理依頼の処理流れ

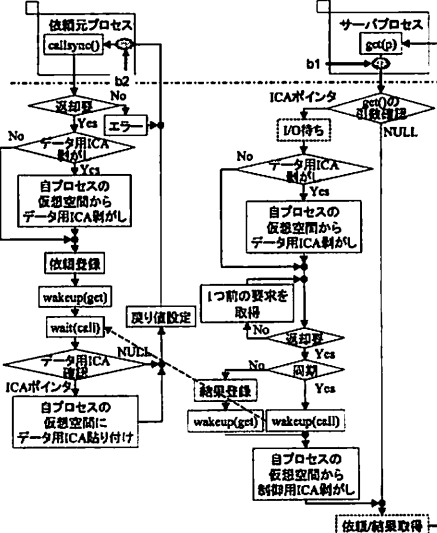


図 5 同期処理依頼における結果返却の処理流れ

表 2 同期処理依頼の処理時間 (単位: μ 秒)

	データ用 ICA なし	データ用 ICA あり	差分
処理依頼	15.47	16.16	0.69
結果返却	11.96	12.77	0.81

3.3 非同期処理依頼

非同期処理依頼における処理依頼と結果返却の処理時間を評価する。非同期処理依頼では、制御用 ICA は依頼元プロセスから剥がし、依頼取得時にサーバプロセスのへ貼り付ける。これは依頼元プロセスによる制御用 ICA 書き換えによるエラーを防止するためである。また、依頼元プロセスは制御用 ICA を保持していないため、結果返却時にはサーバプロセスから制御用 ICA を剥がし依頼元プロセスの結果キューへ登録する必

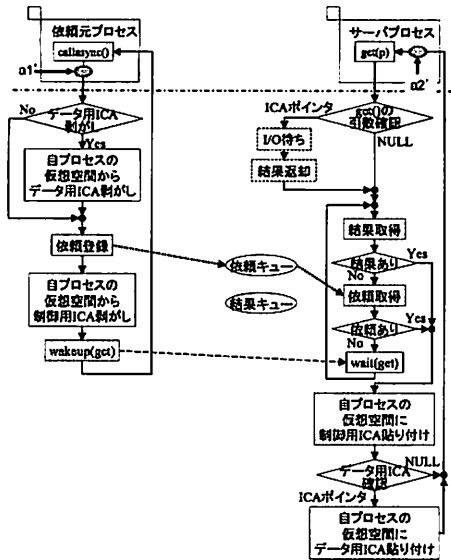


図 6 非同期処理依りの処理流れ

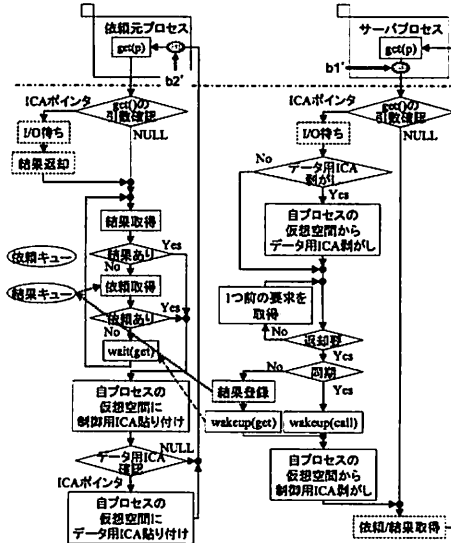


図 7 非同期処理依りにける結果返却の処理流れ

処理依りで $1.15 \mu\text{s}$ 、結果返却で $0.65 \mu\text{s}$ であった。割り込みマスク処理を除いた場合の処理時間の差分は、3.2 節で述べた同期処理依りにける制御用 ICA の貼り替えと結果返却の効率化によるものであると推察する。

4. サービス評価

4.1 項目と環境

サーバプログラム間通信機構のサービス評価として、イーサネットプロトコルサーバと NIC ドライバサーバを介した通信スループットを測定する。具体的には、クライアントサーバシステムを構築し、AnT クライアントと Linux クライアントでイーサネットレベル通信を行い、送信フレームのデータ長ごとのスループットを測定する。測定では、まずクライアントマ

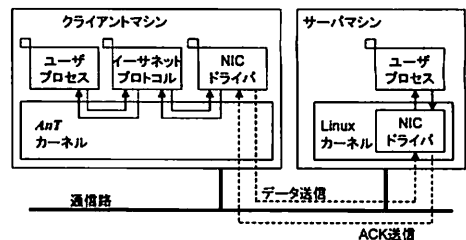


図 8 AnT クライアントにおける測定環境

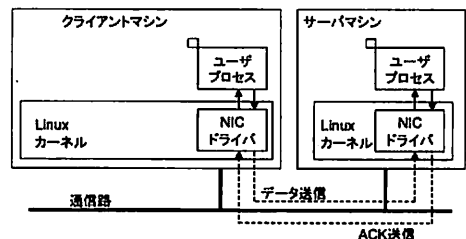


図 9 Linux クライアントにおける測定環境

シがサーバマシンに対してデータを送信し ACK の返信を待つ。サーバマシンはデータを受け取ると ACK として 1 バイトのデータをクライアントへ送信する。クライアントが ACK を受け取った時点で 1 回の送信を完了とする。このデータ送受信を連続で 10000 回繰り返してそのスループットを測定した。なお、クライアントとして Pentium4(R) 2.8GHz プロセッサと RealTek8139 PCI Ethernet Card を搭載した計算機を、サーバとして Pentium(R) Dualcore プロセッサと Intel(R) PRO/100 Network Card を搭載した計算機を利用し、サーバと Linux クライアントには Vine Linux4.2(kernel2.6.16) を利用した。また、測定には RDTSC 命令を利用した。

4.2 通信スループット

4.1 節で示した 2 つのクライアントにおけるそれぞれの測定方法を以下で説明する。

図 8 に示すように、AnT クライアントによる測定では、ユーザプロセス、イーサネットプロトコルサーバと NIC ドライバサーバの 3 つのプロセスを走行させて測定を行う。データの送信を行う場合、まずユーザプロセスがイーサネットプロトコルサーバに依頼を発行し、送信データに対しイーサネットヘッダを加える。次に、イーサネットプロトコルサーバは NIC ドライバサーバに対し依頼を発行し、データの送信を行う。データの受信を行う場合、パケットを受け取った NIC ドライバサーバはイーサネットサーバへ依頼を発行し、受信データからイーサネットヘッダを取り除く。次に、イーサネットサーバはユーザプロセスから受信依頼が発行された際にデータを渡す。パケットの受信より先にユーザプロセスから受信依頼が発行された場合、パケット受信まで依頼を保持し、パケットを受信した際に返すものとする。

また、図 9 に示すように、Linux クライアントによる測定では、AnT クライアントと通信レベルを同等にするために、packet ソケットを利用し、カーネル内のプロトコルスタック処

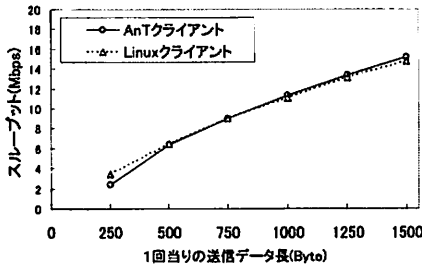


図 10 スループット測定結果

理を通さない低レベル通信を行う。packet ソケットは TCP/IP 等のプロトコルスタックを通さず、直接 NIC ドライバへ通信データを渡すことを可能とする。このため、本測定においてイーサネットヘッダの着脱はユーザプロセスが行う。

測定結果を図 10 に示す。測定結果より、以下のことが分かる。

- (1) **AnT** クライアントにおいて Linux クライアントとほぼ同等の性能が得られていることが確認できる。これより、**AnT** はマイクロカーネル構造を持つため OS 機能をサーバプロセスとして実装しているにもかかわらず、モノリシック構造を持つ Linux と同様な性能を持つことがいえる。
- (2) 送信フレームのデータ長が 750 バイト以下では Linux の方がスループットが高く、750 バイト以上では **AnT** の方がスループットが高い。これは、ICA を利用した複写レスによるデータ転送の影響であると推察する。ICA では、転送データ長が大きい場合、データコピーより ICA 貼り替えの方が高速であるが、転送データ長が小さい場合、ICA 貼り替えよりもデータコピーの方が高速となる性質がある。

5. まとめ

マイクロカーネル OS である **AnT** オペレーティングシステムにおいて、サーバプロセス間での通信を高速に行うためのサーバプログラム間通信機構を評価した。

基本性能の評価から、同期処理では処理依頼に 15.47 μ 秒、結果返却に 11.96 μ 秒、非同期処理では処理依頼に 19.7 μ 秒、結果返却に 18.89 μ 秒要することを明らかにした。また、データ転送にかかる処理時間は、同期処理依頼の場合 4~5%、非同期処理依頼の場合 3~4%であり、ICA によるデータ授受は効率的であることを示した。一方、通信処理の約 80%を割込みマスク処理に要しており、割込みマスク処理効率化の必要性を明確にした。

サービス評価として、ネットワーク通信によるスループットを評価した。評価により、**AnT** はマイクロカーネル構造を持つため OS 機能をサーバプロセスとして実装しているにもかかわらず、モノリシック構造を持つ Linux と同様な性能を持つことを示した。これにより、**AnT** のサーバプログラム間通信機構は効率的であるといえる。また、送信フレームのデータ長 750 バイト前後における **AnT** と Linux の性能逆転は、ICA を利用した複写レスによるデータ転送の影響であると推察する。

残された課題として、割込みマスク処理の効率化と直接返却機能の有効性の評価がある。

謝辞 本研究の一部は、科学研究費補助金基盤研究 (B)「適応性と頑健性を有する基盤ソフトウェアのカーネル開発」(課題番号: 18300010) による。

文 献

- [1] J.Liedtke, 谷口秀夫訳, "真のマイクロカーネルへ向けて," 共立出版, bit, Vol.29, No.8, pp.63-73, 1997.
- [2] Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, "Can we make operating systems reliable and secure?," IEEE Computer Magazine, Vol.39, No.5, pp.44-51, 2006.
- [3] 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," Journal of Information Processing, Vol.14, No.4(19920315), pp.442-453, 1992.
- [4] Andrew S. Tanenbaum, Albert S. Woodhull, "Operating Systems Design And Implementation Third Edition," Prentice Hall, ISBN 0-13-142938-8, 2006.
- [5] Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, "Modular System Programming in MINIX 3," The USENIX Magazine, Vol.31, No.2, pp.19-28, 2006.
- [6] Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, "Roadmap to a Failure-Resilient Operating System," The USENIX Magazine, Vol.32, No.1, pp.14-20, 2007.
- [7] H. Hartig et al., "The Performance of Microkernel-Based Systems," Proc. 16th ACM Symp. Operating System Principles, ACM Press, pp.66-77, 1997.
- [8] 岡本幸大, 谷口秀夫, "**AnT** におけるサーバ間の高速なプログラム間通信機構," マルチメディアと分散処理ワークショップ論文集, pp.61-66, 2007.
- [9] 谷口秀夫, 乃村能成, 田端利宏, 安達俊光, 野村裕佑, 梅本昌典, 仁科匠人, "適応性と堅牢性をあわせ持つ **AnT** オペレーティングシステム," 情報処理学会研究報告, 2006-OS-103, Vol.2006, No.86, pp.71-78, 2006.
- [10] 梅本昌典, 田端利宏, 乃村能成, 谷口秀夫, "**AnT** における高速なプロセス間通信の実現," 第 5 回情報科学技術フォーラム講演文集, pp.135-136, 2006.