AnT におけるサーバ間の高速なプログラム間通信機構

岡本幸大† 谷口秀夫†

AnT オペレーティングシステムは、高い適応性と堅牢性を実現するために、マイクロカーネル構造を持ち、OS 機能の大半を独立した OS サーバとして実現する。このため、OS 処理の際に OS サーバ間の処理連携が多発し、性能が低下してしまう。そこで、この性能低下を抑えるための高速なプログラム間通信機構が求められる。ここでは、高速な呼び出し制御、堅牢性の確保、および、利用にあわせたインタフェース提供を方針としたプログラム間通信機構について説明する。

High-Speed Communication Mechanism between Servers for *AnT*

KOUTA OKAMOTO† and HIDEO TANIGUCHI†

The ***AnT*** operating system is based on microkernel architecture to realize high adaptability and toughness, and it realizes most of OS functions as an independent OS server. Therefore, communication processing between the OS servers occurs frequently in the case of OS processing. Thus, a high-speed communication mechanism between OS servers is required to suppress this deterioration. This paper describes a high-speed communication mechanism between OS servers. That mechanism is based on a design policy, which is *high-speed communication control*, *ensuring of the toughness* and *provision of interface that adapted to the use*.

1. はじめに

計算機が提供するサービスの高度化とともに、計算機の基盤ソフトウェアの機能も高度化し複雑化している。特に、オペレーティングシステム（OS）のプログラム構造は複雑化し、多くの機能を提供している。一方、ひとつの計算機システムを考えた場合、提供するサービスは限られており、OS に必要な機能も限られる。そこで、計算機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多くの資源を有効に活用できる。しかし、必要な機能のみを有する OS を作成するには OS に関する専門的な知識を必要とするため、多くの計算機システムでは、OS 機能の絞込みを放棄している。

OS の機能を計算機システムに比較的簡単に適応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 サーバと名付ける。したがって、必要な機能の絞込みは、OS サーバを生成するか否かにより行うことができる。つまり、比較的簡単に、必要な機能のみを有する OS を作成することができる。Mach[3)] や MINIX3[4)5)6)] では実際にマイクロカーネル構造を採用し、OS の各種機能をプロセスとして実現している。

しかし、マイクロカーネル構造では、OS 機能を OS サーバとして実現するため、OS サーバ間の通信が頻発する。このため、Linux のような一体型 OS に比べ、性能が低下してしまう。

我々は、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[7)](An operating system with adaptability and toughness) を開発している。***AnT*** オペレーティングシステムは、コア間通信データ域[8)] (ICA : Inter-core Communication Area) を有し、プロセス間でのゼロコピー通信の機能を支援している。ここでは、ICA を利用し、OS サーバ間のプログラム間通信を高速に行う機構について述べる。

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

2. *AnT* オペレーティングシステム

2.1 適応性と堅牢性

AnT オペレーティングシステムは、計算機システムの開発者と利用者の両者に「使いやすい」かつ「壊れにくい」という特徴を有する基盤ソフトウェアであることを目指している。

開発者にとって「使いやすい」ためには、新たなサービス（機能）の追加、つまり新たなプログラムの追加を容易にできる必要がある。特に、組込みシステムの場合、新たなドライバプログラムの組込みが頻繁に発生する。このため、ドライバプログラムの新規作成の容易化だけではなく既存プログラムの流用ができれば好ましい。利用者にとって「使いやすい」ためには、高性能であるとともに、利用したいサービスを簡単に利用できる必要がある。したがって、両者の要求を満足するには、基盤ソフトウェアは高い適応性を有することが求められる。

そこで、高い適応性を有する基盤ソフトウェアの構成法が必要になる。具体的には、サービス（機能）の組込みや取外しが簡単なプログラム構成法、および利用者の利用履歴を考慮したプログラム実行制御法が必要である。また、計算機システムの性能低下の大きな要因であるメモリ間データ複写をゼロにする高速なプログラム間データ通信機構も必要である。

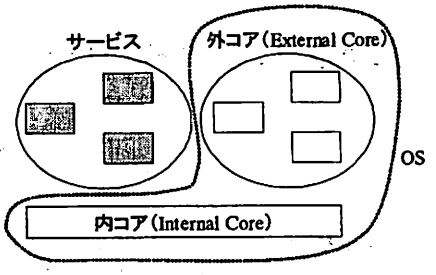
開発者にとって「壊れにくい」ためには、開発中のプログラムの暴走によりシステム全体が壊れることを防ぐ必要がある。利用者にとって「壊れにくい」ためには、いろいろなサービス（機能）を使っても相互干渉によって不都合な動作にならないような高い信頼性とともに、高いセキュリティが必要である。したがって、両者の要求を満足するには、基盤ソフトウェアは高い堅牢性を有することが求められる。

そこで、高い堅牢性を有する基盤ソフトウェアの構成法が必要になる。具体的には、組み込んだプログラムに不具合が発生しても他のプログラムやシステム全体に悪影響を与えない仕組み、およびネットワークを介したセキュリティ低下を防ぐ仕組みが必要である。

2.2 プログラム構造

プログラムは、OS とサービスからなる。OS は、内コアとプロセスとして動作する外コアからなる。サービスは、プロセスからなる。この様子を図 1 に示す。

内コアは、最小のシステムの動作を保証するプログラム部分である．外コアは、適応したシステムに必須なプログラム部分であり、動的に再構成な構造を有する。



(1)OS:内コア＋外コア
・内コア:最小システムの動作を保証する部分
・外コア:適応したシステムに必須な部分(動的再構成構造)
(2)サービス:サービスを提供する部分

図 1　基本構造

サービスは、サービスを提供するプログラム部分である。

2.3 ゼロコピー通信

プロセスと内コアの間あるいはプロセス間の通信を高速化するため、ゼロコピーでのデータ授受機能がある。このためには、大きく以下の 2 つの機能がある。

(1)　ページ単位による領域の確保と解放

(2)　2 仮想空間の間での領域の貼り替え

プロセスと内コアの間あるいはプロセス間で授受を行うデータについては、ページ単位で管理される領域にデータを格納して、通信を行う。この領域をコア間通信データ域（ICA : Inter-core Communication Area）と名付ける。ICA は、内コアにより管理される。仮想空間の様子を図 2 に示す。

プロセスと内コアの間あるいはプロセス間でのデータ授受において、データの複写を避けるため、ICA に格納されたデータは、仮想空間のマッピング表を書き換えることによりデータを授受する。これを ICA の貼り替えと名付ける。ICA はページ単位であるため、このような貼り替えが可能である。

したがって、プロセスが内コアの機能を呼び出す際に渡すデータは、プロセスの仮想空間にある ICA を内コアの仮想空間に貼り替える。一方、内コアがプロセスに実行結果を返却する際に渡すデータは、内コアの仮想空間にある ICA をプロセスの仮想空間に貼り替える。また、プロセス間通信を行う場合、送信プロセスの仮想空間にある ICA を受信プロセスの仮想空間に貼り替える。

つまり、通信する 2 つのプログラム間でのデータ授受を 2 仮想空間の間での ICA の貼り替えにより実現し、ゼロコピー通信を実現する。

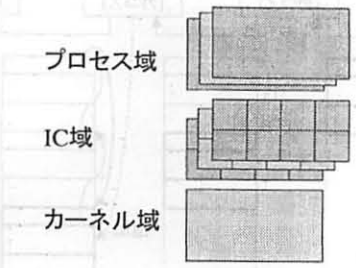


図 2　ゼロコピー通信機能を支える仮想空間

3. プログラム間通信機構

3.1 設計方針と対処法

プログラム間の通信機構を設計するにあたり、高速な呼び出し制御、堅牢性の確保、および利用にあわせたインタフェース提供を方針とした。以降に、各設計方針を説明するとともに対処法についても述べる。

（1） 高速な呼び出し制御

プログラム間の通信、つまりOSサーバから別OSサーバへの手続き呼び出しを高速に行えることが必要である。このために、以下の対処を行う。

(A) ICAを利用して、OSサーバ間でのデータの複写を最小限化する。

(B) 呼び出す手続きの内容に関する情報と扱うデータを別のICAに格納する。ここで、手続きの内容に関する情報を格納したICAを制御用ICAと呼び、扱うデータを格納したICAをデータ用ICAと呼ぶ。これにより、OSサーバ間でのデータの持ち回りを複写レスで行う。

(C) OSサーバの間を次から次に呼び出してゆく多段呼び出しを実現する。

(D) 多段呼び出し時の結果返却を高速化するため、直接返却を実現する。

(2) 堅牢性の確保

OSサーバが多数存在し、それらが連携してOS処理を進める。このため、ひとつのOSサーバの不具合が、他のOSサーバへ影響を与えないようにすること、およびサービスへの影響を最小化することが必要である。このために、以下の対処を行う。

(A) OSサーバは、原則として、処理中のICAをひとつだけ保有する。つまり、呼び出された要求に関する状態保持の量を最小化する。これにより、OSサーバ不具合時の影響を最小化できる。

(B) OSサーバの切り替え機能を実現する。これにより、OSサーバの不具合への対処や機能拡張をサービス継続中に行えるようになる。

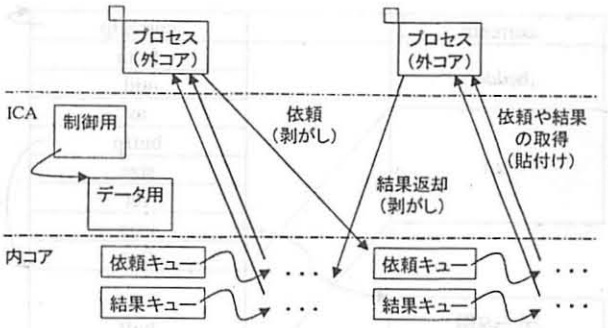


図 3　プログラム間通信の基本機構

（3） 利用にあわせたインタフェース提供

ファイル管理機能や通信処理機能を提供するOSサーバは、多数の処理要求を同時に受け付ける。このため、これらのOSサーバから要求される呼び出しは、非同期型でなくてはならない。一方、非同期型のインタフェース提供は処理が複雑化する。また、そのインタフェースも依頼要求と結果取得に分離されるため、使いやすいとはいえない。これに対し、同期型のインタフェース提供は逆の性質を持つ。そこで、両者のインタフェースを選択して利用できる機構を実現する。

3.2 基本構造

プログラム間の呼び出し制御の基本機構を図3に示す。内コアは、各OSサーバに依頼キューと結果キューを用意している。呼び出しは、制御用ICAとデータ用ICAを授受（仮想空間上の剥がしと貼り付け）することで行う。基本的な呼び出しの処理を以下に説明する。

（1） 依頼元プロセスが依頼先プロセスの依頼キューに依頼を登録する。

（2） 登録された情報を依頼先プロセスが取得し処理を実行する。

（3） 依頼先プロセスが依頼元プロセスの結果キューに結果を登録して返却する。

次に、制御用ICAの形式を図4と表1に示す。制御用ICAは、手続きの内容に関する情報を格納しており、依頼元や依頼先に関する情報も持つ。また、データ用ICAへのポインタ情報を持つ。これにより、OSサーバ間でのデータの持ち回りを複写レスで行うことができる。

たとえば、ファイル管理、バッファキャッシュ制御、ディスク(DK)ドライバの3つのOSサーバ間でのデータ入出力を考える。データ入力の場合、DKドライバで入力されたデータはデータ用ICAに格納され、バッファキャッシュ制御ではキャッシュ管理配下で管理され、ファイル管理を経由して応用プログラム（AP）

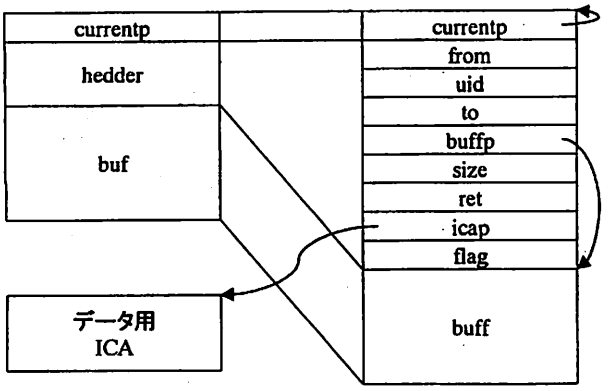


図 4　制御用 ICA の形式

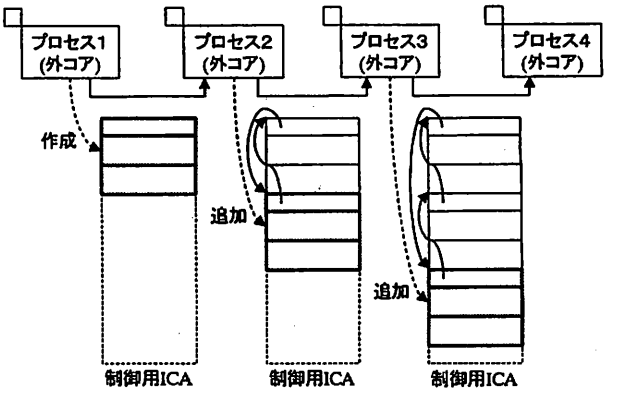


図 5　制御データスタック

表 1　制御用 ICA のヘッダ情報

エントリ名	内容
currentp	現在の制御データへのポインタ
from	依頼元のプロセス情報
uid	依頼元の利用者情報
to	依頼先のプロセス情報
buffp	パラメータ域の先頭へのポインタ
size	パラメータ域の大きさ
ret	返却値
icap	データ用 ICA へのポインタ
flag	同期/非同期，返却の要/非， I/O 待ちの要/非，データ用 ICA 剥がしの要/非

に渡される。これら一連の処理をデータ複写レスで行うことができる。データ出力の場合も同様である。

なお、currentp や flag は、多段呼び出しや直接返却などを実現するための情報である。詳細は、3.3 節で説明する。

3.3　特徴的な構造

3.3.1　多段呼び出しと直接返却

OS サーバの連携のため、OS サーバの間を次から次に呼び出してゆく多段呼び出しを効率的に実現する必要がある。呼び出す度に新たな制御用 ICA を確保する方法は、処理オーバヘッドが大きくなるため、ひとつの制御用 ICA に多段呼び出しに関する情報を積み重ねることとした。この様子を図 5 に示す。OS サーバの間を次から次に呼び出すに伴い、手続きの内容に関する情報を同じ制御用 ICA に積み重ねる。これに伴い、受け取った OS サーバが依頼された情報を把握できるように、制御用 ICA の先頭に最新の情報格納域へのポインタ（currentp）を持つ。制御用 ICA の先頭にない currentp は一つ前の呼び出し情報格納域へのポインタを持つ。このように、currentp により、積み重ねられた情報をたどれるようにする。

次に、多段呼び出し時の結果返却を高速化するための直接返却について述べる。この様子を図 6 に示す。

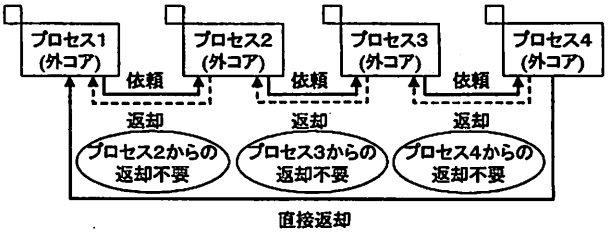


図 6　直接返却

多段呼び出しされた処理は、必ずしも逐次返却が必要ではない。たとえば、AP プロセスから要求されたデータ入力は、ファイル管理、バッファキャッシュ制御、DK ドライバの 3 つの OS サーバを介して処理が進められる。しかし、入力データの結果返却は、DK ドライバから直接に AP プロセスに返却できる場合もある。これを可能にするため、制御用 ICA に flag を設け、返却の要否を設定できることとした。currentp と flag を利用して、直接返却が可能である。

3.3.2　保持状態の最小化

計算機システムの堅牢性を確保するため、OS サーバが保有する保持状態を最小化する。このために、以下の対処を行う。

（1）　ICA 貼り替え契機の工夫

非同期処理では、制御用 ICA を依頼時に依頼元プロセスから剥がし、依頼取得時に依頼先プロセスへ貼り付ける。これにより、依頼元プロセスの制御データの書き換えによるエラーを防止する。また、同期処理では、制御用 ICA を依頼元プロセスに貼り付けたままとし、依頼取得時に依頼先プロセスへ貼り付ける。これにより、貼り替え処理を削減でき、同期処理を効率化できる。なお、依頼元プロセスは wait 状態のため制御データ書き換えによるエラーは発生しない。

（2）　制御用 ICA の保持

OS サーバは、一度に 1 つの制御用 ICA しか保持しないこととする。これにより、OS サーバ異常終了時の

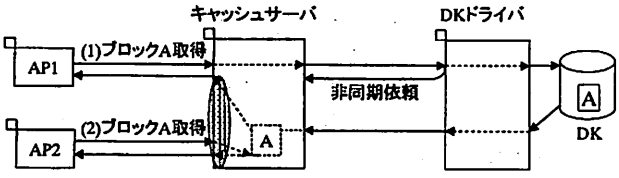


図 7　制御用 ICA の複数保持

影響を抑制するとともに、OS サーバ入れ替えを可能にしている。ただし、キャッシュ機能を持つ OS サーバの場合、複数の制御用 ICA を保持する。この様子を図 7 に示し、以下に説明する。

(A)　AP1 がキャッシュサーバにブロック A の取得を依頼する。

(B)　ブロック A はキャッシュに存在しないため、DK ドライバにブロック A の取得を非同期で依頼し、処理終了を待たずに戻る。

(C)　AP2 がキャッシュサーバにブロック A の取得を依頼する。

(D)　ブロック A はキャッシュ予定なので DK ドライバから返却されるまで待つ。

(E)　DK ドライバからブロック A が返却され、AP1 と AP2 にそれぞれブロック A を返却する。

3.3.3　非同期・同期の処理

利用にあわせたインタフェース提供として、非同期処理と同期処理の両インタフェースを提供する。

OS サーバには複数の AP プロセスから処理の要求が依頼される。このため、OS サーバが依頼された処理実行のために別サーバに処理を依頼する場合は、非同期処理インタフェースでなくてはならない。さもないと、別サーバへの処理依頼により、新しい要求を受け付けることができない。このときの処理の流れを図 8 に示し、以下に説明する。

(1)　依頼元プロセスは、依頼先プロセスの依頼キューに依頼を登録し、依頼先プロセスの処理終了を待たずに依頼から復帰する。

(2)　依頼先プロセスの処理結果は、依頼元プロセスの結果キューに登録する。

(3)　依頼元プロセスは、自プロセスのキューを確認し結果を取得する。なお、処理依頼よりも結果の返却を優先するために、結果キューからの結果取得を先に行い、結果キューがなければ依頼キューからの依頼取得を試みる。

非同期処理と同期処理の利用にあわせた発行契機の例を以下に示す。

(1)　AP プロセスから OS サーバへの依頼は同期とする。この要求は、システムコールに相当するためである。なお、システムに精通した高度な AP は、非同期も利用できる。

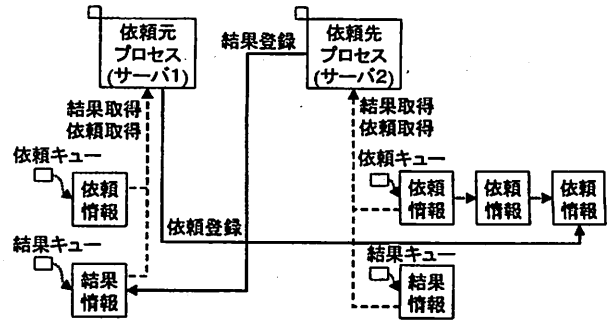


図 8　非同期処理の流れ

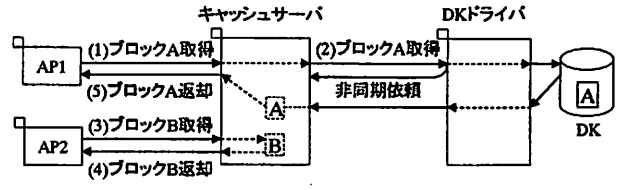


図 9　非同期による DK ドライバの処理例

(2)　OS サーバから OS サーバへの依頼は非同期とする。これは、OS サーバが複数の要求を同時に処理するためである。

(3)　入出力待ちが発生するドライバプロセス（OS サーバのひとつ）への要求は、同期とする。これは、OS サーバ内の保持状態の最小化、および OS サーバ内処理の簡易化のためである。

なお、キャッシュ機能を持つ OS サーバからのドライバプロセスへの要求は、非同期である。これは、キャッシュ機能を持つ OS サーバはキャッシュを保持しているので、DK リード中に既にキャッシュ中にあるデータの読み込み依頼を返却するためである。この様子を図 9 に示し以下に説明する。

(1)　AP1 がキャッシュサーバにブロック A の取得を依頼する。

(2)　ブロック A はキャッシュに存在しないため、DK ドライバにブロック A の取得を非同期で依頼し、処理終了を待たずに戻る。

(3)　AP2 がキャッシュサーバにブロック B の取得を依頼する。

(4)　ブロック B はキャッシュに存在するため、AP2 にブロック B を返却する。

(5)　DK ドライバからブロック A が返却され、AP1 にブロック A を返却する。

4.　評　　価

本機構を *AnT* 上に実装し、Pentium4 プロセッサ

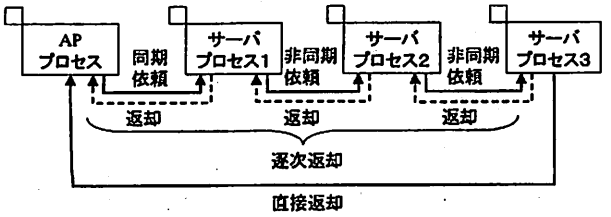


図 10 評価の様子

2.8GHz を使用して測定を行った。また、処理時間の計測には RDTSC 命令を使用した。本測定の様子を図 10 に示す。AP プロセスが依頼を発行してから結果が返却されるまでの時間を、逐次返却の場合、直接返却の場合それぞれで測定を行った。なお、各プロセスは依頼取得、および結果返却以外の処理は行わないものとする。

測定の結果、逐次返却の場合 154707 クロック、直接返却の場合 103992 クロックかかっており、直接返却を行うことにより 50715 クロック (約 33%) の速度向上が見られた。また、これは返却処理 2 回分のオーバヘッドの削減であり、1 回の返却処理の省略につき約 25000 クロック (約 15%) のオーバヘッドの削減が見込める。

5. おわりに

マイクロカーネル構造を持ち、OS 機能の大半をサーバとして実現している ***AnT*** オペレーティングシステムについて、サーバ間の高速なプログラム間通信機構について述べた。

高速なプログラム通信機構の設計方針として、高速な呼び出し制御、堅牢性の確保、および利用にあわせたインタフェース提供について説明した。これらの方針を受け、ICA を利用し、かつ制御用とデータ用の ICA を分離し、データの複写を最小限とした。また、複数のサーバにまたがる処理を高速化するため、多段呼び出しと直接返却を可能にする制御機構を示した。また、サーバが保有する保持状態を最小化することで、堅牢性を確保した。さらに、利用の形態にあわせ、非同期と同期のインタフェースを提供する。

評価結果として、直接返却を用いることにより、一回の返却処理の省略につき約 15%のオーバヘッド削減が見込めることが分かった。

残されて課題として、サーバプログラム間通信の高速化、および応用プログラムでの評価がある。

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。

参考文献

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) 谷口秀夫, 乃村能成, 田端利宏, 安達俊光, 野村裕佑, 梅本昌典, 仁科匡人, "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).
8) 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論文集, pp.135-136(2006).