システムソフトウェアと
オペレーティング・システム 66－3
（１９９４．９．１６）

次世代アーキテクチャ向けオペレーティングシステムの開発 プロトタイプの性能評価

岡本 利夫* 津田 悦幸 福本 淳 寺本 圭一

(株) 東芝 研究開発センター 情報・通信システム研究所

我々が現在設計中の OS は、広大な 64 ビット・アドレス空間を利用した単一仮想記憶空間のモデルを導入し、アクセス制御リストに基づくページ単位のメモリ保護機構により、プロセス間通信をサブルーチンコールのように高速にかつ安全に実行でき、OS の通信コストを低下させることが目標である。本報告では、本 OS モデルの機能検証を目的に試作した、既存の 32 ビットプロセッサ版のマイクロカーネルで、プロセス間通信の速度を測定した。その結果、Mach や Unix のプロセス間通信の速度より高速で、上記目標を達成していることが確認できた。

Estimation of a Trial Implementation of an Operating System for Next Generation Architecture

Toshio OKAMOTO, Yoshiyuki TSUDA, Atsushi FUKUMOTO and Keiichi TERAMOTO

Communication & Information Systems Research Lab.
R&D Center
TOSHIBA CORPORATION

In our newly designed operating system (OS), a fast and secure inter process communication (IPC) just like a normal subroutine call and low overhead IPC mechanism are offered, using a huge 64-bit addressable SVS (Single Virtual Space) model and a page-size-grained memory protection based on access control lists. To estimate the functional satisfactions of our OS model, we have just developed a microkernel-based operating system prototype on a 32-bit CPU.

This paper describes the estimation of our trial implementation, focusing on the IPC speed as comparison with major OS: Unix and Mach. According to the measurements, these features were confirmed by the result that the speed of our prototype's IPC was faster than these OS's.

*email: oka@isl.rdc.toshiba.co.jp

1 はじめに

現在、標準OSとして広く普及しているUnix[1]は、様々な機能が加えられ発達してきたが、基本設計が20年以上も前になされたもののため、新しいハードウェア、ソフトウェアモデルに必ずしも合わなくなってきてきる。

我々は、既報[2, 3, 4]の通り、新しいディジタルメディア時代に適合した広い範囲に適用する新しいオペレーティング・システムCubix (CUBe of 2 byte unIX)[2]を研究している。

Cubixは64ビットプロセッサでの利用を想定し、単一の広大なアドレス空間(2^{64} = 16エクサバイト)を積極的に利用し、かつ、プログラム(Unixでいうところのプロセス)とファイルを区別することなく、その空間中にすべて配置するメモリモデルが特徴である。このモデルによって、従来システムコールを介して行われるIPC(プログラム間通信)やファイルアクセス(read, writeなど)を、Cubixではメモリアクセスの形で直接行なえるので、OSのオーバヘッドが低下し、高速なプログラム実行を可能とする場が提供できる。

また、従来プログラムごとに独立な仮想記憶空間を割当て、空間の独立性を利用してプログラムを保護してきたが、Cubixでは空間とは独立した保護概念を導入し、同一アドレス空間内でもメモリ保護を可能とした。

前回の報告[5]において、Cubixモデルの機能検証を行う目的で、実機のi386/486を用いた32ビットプロセッサマシン(J3100, Dynabook)の上にCubixプロトタイプ(以降Cubix386と称す)を試作したことを述べた。

今回は、試作したOSにおける評価に関して報告する。

2 Cubixの概要

Cubixシステムの特徴を、以下に簡単にまとめる。

1. 単一仮想記憶空間 (SVS:Single Virtual Space)

   Unixなどの OSでは、プログラムごとに仮想記憶空間が割り当てられ、互いに空間が独立している多重仮想記憶空間(MVS:Multi Virtual Space)方式であるが、Cubixではすべてのプログラムやデータが共通の単一仮想記憶空間中に存在する(図1)。

   このSVSの導入により、プログラム間の通信は、メモリポインタの受け渡しとサブルーチンコールで行えるので、大幅な通信速度の向上が期待できる。

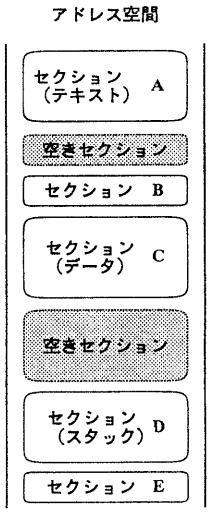

図1: 単一仮想記憶空間構成概念図

2. 一元化記憶域 (OLS:One Level Storage)

   ディスク上のデータ(ファイル)とメモリ中のデータを区別なくアクセスできるように、すべてを仮想記憶空間中に存在させ、ファイルアクセスを高速に行うことを可能にする。

3. 高機能メモリ保護機構を用いたアクセス保護

   1つの仮想空間は、メモリセクションと呼ぶ、ページを単位とする領域に分割されている。各メモリセクションには「スレッドID」と「テキストセクションID」(現在スレッドが実行しているプログラムの存在するメモリセクションID)の2つをキーとしたアクセスコントロールリスト(ACL)を使ってアクセス保護を設定できる。これにより、SVSでもMVS同様のアクセス保護を実現できる。

[1] UNIX is a registered trademark licensed exclusively to X/Open Company, Ltd.

[2] 2バイトマシンであるPDP11上で最初にUnixが開発されてから3世代目のUnixという意味も込めている。

ページ単位にメモリセクション領域を設定するのは、

- アクセス保護の設定管理は OS
- アクセス保護の実行は MMU（ハード）

として既存のメモリ管理機能との整合性をはかり、オーバーベッドの問題を解決するためである [1]。

4. マイクロカーネルアーキテクチャ

Cubix は、マイクロカーネルアーキテクチャ構成を採用している。したがって、Cubix マイクロカーネルの上に Unix サーバなどの各種システムサーバがのせられ、柔軟な OS の構成ができる。この形で既存の各種 OS との互換性を提供できる。

また、スケジューラやページャなどのシステムサーバも、マイクロカーネル外に実装している。

Cubix システムの全体構成を図 2に示す。

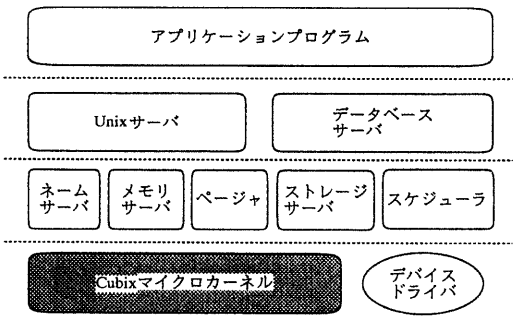

図 2: Cubix 構成概念図

3 IPC の性能評価

3.1 単一仮想空間における IPC の特徴

図 3は、Unix に代表される従来型 (MVS) による IPC である。各ユーザプログラムは、別々の仮想空間に割り当てられ、空間の違いでユーザプログラムの保護を行っている。したがって、他のプログラムに処理を依頼する必要があると（例えば、クライアントプログラムがサーバプログラムに処理を依頼する場合）、OS が提供するサービス（システムコール）で仲介してプログラムの実行を依頼する。その結果、コンテクストスイッチが発生し、プログラムの仮想空間が切り替わる。処理が終了すると再度、元の仮想空間に切り替わる。

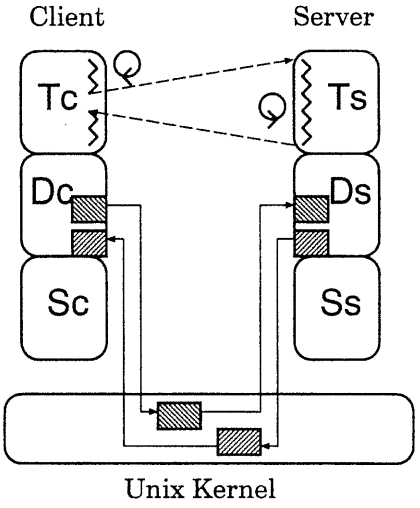

図 3: UNIX モデルにおける IPC

通常、ユーザプログラムと OS とは同一仮想空間に存在するが、OS の資源を守るため、OS の実行は保護属性が異なるカーネルモードでしか実行できない。ユーザモードからカーネルモードへ切り替えるのがシステムコールであり、システムコールによって、コンテクストスイッチの他に実行モードの切り替えも発生している。

他のプログラムへ実行の依頼時には、仮想空間が異なり他の空間から参照できないので、パラメータ等の実行に必要なデータも、OS 経由でコピーされる。

以上、クライアント・サーバプログラム間の処理中に、コンテクストスイッチが 2 回、実行モードの切り替えが 2 回、パラメータのコピーが 4 回発生する。[3]

一方、図 4は、Cubix における単一記憶空間における IPC である。

サーバプログラムとクライアントプログラムが同一空間に存在し、1 つのスレッドが両者の間を飛び

[3] もちろん、シェアドメモリ、Copy on Write などの技術を利用すれば、メモリのコピーを減少できる。

回って一連の処理を行う。

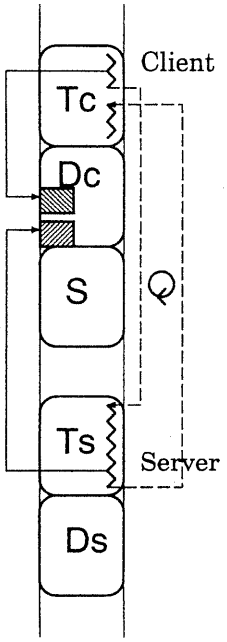

図 4: Cubix モデルによる IPC

この場合、図から明らかなように、領域保護の問題をのぞけば、

- アドレスポインタによる処理先の指定
- アドレスポインタによるパラメータの指定 (コピー不要)
- アドレスポインタによるスタックの共有

などにより、通常のサブルーチンコールのように IPC を実行できる。つまり、クライントのスレッドをサーバのアドレス空間中で直接実行させて高速化をはかる LRPC[10] を SVS に拡張したことと同等である。

この場合のメリットとして、

1. IPC に伴うパラメータのコピーが不要
2. 仮想空間の切りが替え不要なため、コンテクストスイッチに伴う空間管理用のデータの退避・復旧が不要
3. IPC ごとの実行モードの切り替えが不要
4. 仮想空間の切り替えが不要であることによる、論理キャッシュのヒット率の向上
5. 同一スレッドを長期間利用し、種々のサーバへ IPC を繰り返す利用形態をとることによるスレッド管理コストの減少

などがあげられる。

近年の RISC プロセッサにおける速度向上に比べて、メモリのアクセス速度は向上しておらず、メモリコピーのオーバヘッドは、無視できない大きさになってきた。また、RISC プロセッサにおける速度向上は、もっぱら数値演算などのアプリケーションの処理性能に寄与し、トラップ、コンテクストスイッチ、システムコールなどの OS 処理性能は、それほど向上してないという指摘 [11] がある。上記の特徴は、これら指摘の問題点の有力な解決策になる。

しかし、従来の MVS と同様な領域保護機能が必要であり、また、そのオーバヘッドが少ない程、有利となるわけで、そのハード、ソフト機構の検討とオーバヘッドの評価は重要である。

3.2 Cubix における IPC の実装方法

Cubix おける IPC は、次の 2 つの方法を提供している。1 つは、上記で説明したように、クライアントプログラムで実行中のスレッドがサーバプログラムを直接実行するもので、EPC と呼んでいる。もう 1 つは、従来と同様、クライアントとサーバの両方にスレッドを配し、スレッド間でデータをやりとりして処理を行う形態で、ITC と呼んでいる。

EPC (External Procedure Call, 外部手続き呼出し)

スレッドを切替えず、同一スレッドを他のメモリセクションへジャンプさせることによって他のプログラムを実行する機構である。(図 5)

この方式の特徴は、

- 同一スレッドを用い、スケジューラを介在させないので、非常に処理が軽い
- SVS の特徴を生かし、両プログラム間の引き数をポインタ渡しで行える他、スタックも共用するので、引き数や返り値をスタック渡しで行う事も可能

である。

しかし、

- サーバがクライアントのスタックを破壊する可能性がある

のが欠点である。

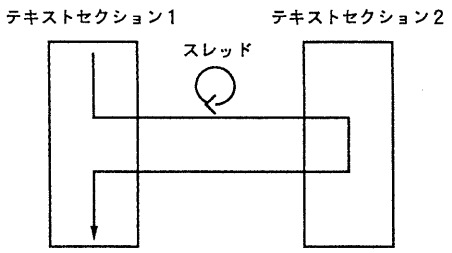

図 5: EPC 方式による IPC

ITC (Inter-Thread Communication, スレッド間通信)

独立したスレッド同士が、スケジューラが提供する同期機構と共有メモリとを使って通信する機構である。Unix 等の通常の OS での RPC に近い。(図 6)

- 外部スケジューラを介す点とスレッドを切り替える点で、EPC よりも重い

という欠点があるが、

- 通信データをコピーせず共有メモリでポインタを利用してやりとりする分だけ通常の RPC より軽い
- スレッドを切り替えたり、スタックを共用しないので、サーバプログラムの暴走などにより、クライアントが破壊されることがなく、EPC より安全である

といった特徴がある。

両者は、アプリケーションプログラムの特性によって使い分ける。

3.3 Cubix における IPC の測定

IPC の測定は、Cubix, Unix, Mach[9] の 3 つの OS を対象にいくつかの IPC の条件を設定して行った。

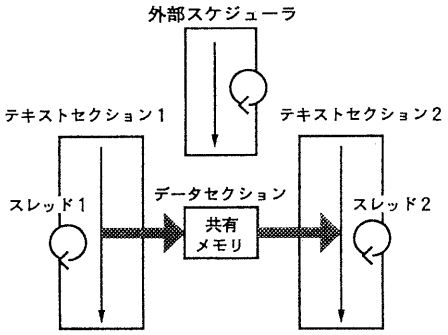

図 6: ITC 方式による IPC

1. IPC のモデルパターン

- <同期, 非同期通信>
- <引数, 戻り値の組合せ>
  - 引数なし, 戻り値なし (NullProc)
  - 引数あり, 戻り値なし (ArgNoRtn)
  - 引数あり, 戻り値あり
- <引数処理の仕方 (引数, 戻り値あり) >
  - サーバ内で予め用意した戻り値を返す (ArgRtn_NoCopy)
  - 戻り値として引数をコピーしたものを返す (ArgRtn_Copy)
- <引数の大きさ>
  - 2 Kバイト
  - 4 Kバイト
  - 8 Kバイト

この測定モデルは、デジタルメディア時代のサーバ・クライアントモデルで処理することを念頭に置いている。つまり、引数・戻り値として比較的大きなデータを与えている。また、引数処理の仕方の違いは、サーバ内でのデータ処理の影響をのぞいた場合と、サーバ内で多少のデータ処理[4]を行い、総合的な IPC の実行速度を反映するようにしたものである。

[4] たとえば、送られてきたイメージデータをカラーマップテーブルによって色変換処理を行うこと

2. プロセス間通信機能の特徴

測定に使用した各 OS におけるプロセス間通信機能を以下に挙げ、特徴を簡単にふれる。

- Mach2.5, 3.0 – IPC (MIG 使用)
  - 同期, 非同期通信可能 (非同期時の戻り値は不可 (エラーコードは返す))
  - 複数の引数および複数の結果を受渡し可能
- BSD/386 1.1 – Sun RPC(rpcgen 使用)
  - 同期通信のみ
  - 単一の引数および結果のみ可能
  - 引数には構造体を指定でき、データタイプの互換性をとる (XDR)
  - トランスポート・プロトコルには udp と tcp が可能 (今回は udp を使用)
- BSD/386 1.1 – Socket
  - Socket を直に利用して簡単な同期通信機構を作成
  - 単一の引数および結果のみ可能
  - トランスポート・プロトコルには udp と tcp が可能 (今回は udp を使用) [5]

3. 測定環境測定に利用した条件は、表に示す通りである。

マシン		J-3100 PV (486DX2-66)
メモリ		12M バイト
OS	Cubix	Cubix386
	Mach	Mach2.5
		Mach3.0
	UNIX	BSDI1.1

測定は、Mach, Unix は、ライブラリ関数 (gettimeofday) により測定。Cubix は、タイマークロックのカウンタ (1.193MHz) を利用して測定している。

Cubix の IPC(EPC) に関しては、独自のメモリ保護機能を実現するのに、ハード化した場合 (EPC-H)[1][6] と現状のハードのままでソフトでエミュレートした場合 (EPC-S) と両方測定した。

[5] Unix ドメインによる通信方法もあるが今回は利用しなかった

[6] 100%ACL がヒットすると仮定して測定している

3.4 Cubix における IPC の測定結果

今回の報告ではポイントを絞った結果のみを述べる。

条件は、8 K バイトのデータを引き数・戻り値にもつ同期通信にて、4 つのモデル (ArgRtn_Copy, ArgRtn_NoCopy, ArgNoRtn, NullProc) にて、測定結果を以下の表に挙げる。(†は測定限界以下である。)

8K バイトの大きさは、ページ単位の大きさのデータを送ることを想定している。

ArgRtn_Copy		
IPC	Type	Time(msec)
Cubix386	EPC-S:	1.07
	EPC-H:	0.62
	ITC:	1.89
Mach	2.5:	3.28
	3.0:	3.27
Sun-RPC		31.3
UDP-Socket		5.53

ArgRtn_NoCopy		
IPC	Type	Time(msec)
Cubix386	EPC-S:	0.163
	EPC-H:	0.001†
	ITC:	0.902
Mach	2.5:	2.02
	3.0:	1.97
Sun-RPC		31.0
UDP-Socket		5.30

ArgNoRtn		
IPC	Type	Time(msec)
Cubix386	EPC-S:	0.163
	EPC-H:	0.001†
	ITC:	0.902
Mach	2.5:	1.21
	3.0:	1.09
Sun-RPC		17.1
UDP-Socket		3.18

NullProc		
IPC	Type	Time(msec)
Cubix386	EPC-S:	0.163
	EPC-H:	0.001†
	ITC:	0.902
Mach	2.5:	0.229
	3.0:	0.106
Sun-RPC		1.46
UDP-Socket		0.953

全体的に見て、Sun-RPC は、引き数・戻り値のデータタイプをチェックして、CPU のタイプに合わせて変換する XDR の処理がかなり重い。

データ変換処理をしない残りの方式で比べると、Cubix の方が性能が良い。表によれば、画像通信のような比較的大きなデータを使うサーバを想定した場合 (ArgRtn_Copy)、メモリーコピーのオーバヘッドの影響が大きく、Cubix の ACL によるアクセス保護をハードで実現した場合の外部手続き呼出しの場合 (EPC-H)、非常に高速にサーバクライアントシステムを構成できることがわかる。また、上記アクセス保護をソフトで実現した場合 (EPC-S) も、Unix の約 5 倍、Mach の約 3 倍高速であることがわかる。ITC でも、後に述べるように、ITC 自身の性能は良くないが、総合的には、Unix、Mach より性能が良い。

もちろん、サーバ内の処理を抜いた IPC の往復部分の通信性能のみで比較した場合 (ArgRtn_NoCopy)、ポインタ操作で引き数・戻り値が渡せる EPC でその差は拡大する。

NullProc、つまり、データなしの IPC の通信部分のみの速度で比較すると、SVS の Cubix はコンテクストスイッチ（仮想空間の入れ替え）をしない分、高速であるはずであるが、現バージョンの Cubix ではそれほど速くはない。EPC では、Mach と同程度である。これは、Mach の IPC が Mach3.0 で最適化されていることと同時に[7]、Cubix のメモリセクションがセグメント機構を使って実現しているため、セグメントの切り替えコストが高いことも原因であると考えられる。ITC で比較した場合、スケジューラがカーネル外にあるため、スレッド切り替えが頻繁に発生し、性能が良くない。[8]

[7] もちろん、Cubix386 が機能試作を目的に作られたので、コードが最適化されてない点も大きい

[8] 1 回の ITC で、スケジューラが 4 回呼ばれるので、性能劣化が大きい

4 おわりに

以上、Cubix386 マイクロカーネルのプロトタイプの評価結果について報告した。現在、細かい評価、特に ACL の評価に関しては継続して行っており、別の機会に報告したい。

Cubix386 システムは、マイクロカーネル部の開発は既に終了し、マルチスレッドで動作をはじめている。また、外部スケジューラからのスレッドコントロール、外部ページャによる仮想メモリシステム、簡易なファイルシステムの実装が終わっている。

今後の開発は、本評価結果を反映させ、実用をめざした実装を予定している。

近年のネットワークや共有メモリ型マルチプロセッサの発達にともない、分散指向の OS が数多く登場している。Cubix の研究もマイクロカーネルによるネットワーク透過な分散システムに発展させていきたい [8]。

参考文献

[1] 野末 浩志、斎藤 光男, 『大規模アドレス空間内でのデータ保護方式の実現』, 情報処理学会第 44 回大会, 4D-7, 1992 年 3 月.

[2] 申 承昊、瀬川 英生、野末 浩志、岡本 利夫、前田 賢一、斎藤 光男, 『大規模アドレス空間を利用する OS の構想』, 情報処理学会第 44 回大会, 1G-5, 1992 年 3 月.

[3] 瀬川 英生、野末 浩志、申 承昊、岡本 利夫、前田 賢一、斎藤 光男, 『単一仮想記憶域を提供する 64 ビット・アドレス指向 OS』, 情報処理学会 研究報告 92-OS-55, 1992 年 6 月.

[4] 瀬川 英生、申 承昊、野末 浩志、高橋 俊成、岡本 利夫、前田 賢一、斎藤 光男, 『単一仮想記憶域を提供する 64 ビット・アドレス指向 OS』, 情報処理学会 第 4 回コンピュータシンポジウム, 情処シンポジウム論文集 Vol. 92, No. 7, 1992 年 10 月.

[5] 岡本 利夫、申 承昊、福本 淳、吉田 英樹, 『次世代アーキテクチャ向けオペレーティングシステムの開発: プロトタイプの作成』, 情報処理学会 第 5 回コンピュータシステム・シンポジウム, 1993 年 10 月.

[6] 津田 悦幸、福本 淳、寺本 圭一、友田 一郎、岡本 利夫, 『次世代アーキテクチャ向けオペレーティングシステム: マイクロカーネルの評価』, 情報処理学会 第 49 回全国大会, 1994 年 9 月.

[7] 友田 一郎、津田 悦幸、岡本 利夫, 『64bit 単一仮想記憶 OS における X Window System のクライアント／サーバー通信機構』, 情報処理学会 第 49 回全国大会, 1994 年 9 月.

[8] Toshio Okamoto, Hideo Segawa, Sung Ho Shin, Hiroshi Nozue, Ken-ichi Maeda, Mitsuo Saito, "A Micro Kernel Architecture for Next Generation Processors," *Proceedings of the USENIX workshop on Micro-kernels and Other Kernel Architectures,* April 1992.

[9] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, "Mach: A New Kernel Foundation For UNIX Development," *Proceedings of the USENIX 1986 Summer Conference,* pp.93-112, 1986.

[10] Brian N. Bershad, Thomas Anderson, Edward Lazowska and Henry Levy, "User-Level Interprocess Communication for Shared Memory Multiprocessors," *ACM Transactions on Computer Systems,* Vol.9, No.2, pp.175–198, May 1991.

[11] Thomas E. Anderson, Henry M. Levy, Brian N. Bershad, and Edward Lazowska, "The Interaction of Architecture and Operating System Design," *ASPLOS-IV,* pp.108–120, April 1991.