オペレーティング・システム 51−12
（1991. 7. 18）

オブジェクト指向オペレーティングシステムの評価

平瀬 吉也　伊豆田 和也　大久保 英嗣　白川 洋充　大野 豊

立命館大学理工学部情報工学科

分散オペレーティングシステム Theta(Threads and Tasks) は，分散メモリ型の並列計算機をサポートする目的で開発された．一般にオペレーティングシステムは多種多様なデータ構造とプロシジャが存在し，複雑化と肥大化を招いている．そのためシステムの記述が複雑で困難を伴う．そこでシステムの効率の良い記述を行うためにオブジェクト指向の技法を用い，システムの抽象化を行うことによりその記述性·生産性を向上させた．今回，C++やIBM 東京基礎研究所で開発されたCOB(C with OBjects) を用いてオブジェクト指向の技法を反映させ Theta を記述し，C との性能の比較を行った．本論文ではその評価結果について述べる．

Evaluation of an Operating System using Object-Oriented Techniques

Yoshiya Hirase　Kazuya Izuta　Eiji Okubo　Hiromitsu Shirakawa
Yutaka Ohno

Department of Computer Science and Systems Engineering, Faculty of Science and Engineering, Ritsumeikan University

56-1 Toujiin kita-machi, Kita-ku, Kyoto 603, Japan

The distributed operating system Theta has been developed with the aim to support applications on concurrent computers with distributed memory. Generally, operating systems handle many and various kinds of data, so their description has become complex and difficult. Using object oriented techniques and, by the abstraction of the system, their description and production are improved. The efficiency of a new version of Theta described by C++ and COB, with object oriented techniques, has been compared with its previous C version. In this paper, the evaluation results of such comparison are stated.

1 はじめに

マイクロプロセッサの高機能化，低価格化が進んでいる現在，研究や実際のアプリケーションのために並列·分散処理を適用する要求が高まっている．このため，分散オペレーティングシステム(OS)の研究が盛んに行われている．この中で，Thetaは，分散メモリ型の並列計算機をサポートする目的で開発されたOSである．今回，我々は，オブジェクト指向型言語であるC++とCOB[5]を用いてThetaの再記述を行い，Thetaにオブジェクト指向の技法を反映させた．これは，システム構成要素をクラス化し，継承を利用してシステムを構造化することによって，OS自体のコードの共有·再利用を推進することを目的としている．本論文では，オブジェクト指向型言語C++とCOBで再記述を行ったThetaと，Cで記述したThetaとの性能を比較しその評価を行った．

以下，2章ではThetaの概要を述べ，3章ではThetaに適用したオブジェクト指向の技法について述べる．また，4章では各言語の特性を挙げ，5章ではThetaにオブジェクト指向の技法を用いた場合の性能の評価に関して述べる．

2 分散OS Thetaの概要

Thetaは，研究用の実験的な分散メモリ型の並列計算機上で並列·分散システムを効率良く稼働させるためのOSとして開発された．Thetaはターゲットマシンとして VMEベースのMC68030 CPUボードを採用し，システムの開発はSun3/60またはSPARC station1+で行っている．

以下，本章では，Thetaの特徴について述べる．

2.1 極小カーネル

Thetaでは，カーネルの機能をコンテキストの切替えのみとしている．従来のシステムの機能はカーネルの外に置き，システムサーバとライブラリ関数により実現した．このようにOSのほとんどの機能をシステムサーバで実現することにより，OS自体の並行性が向上し，その結果として，システム全体の並行性を高めることができる．

カーネルの機能はコンテキスト切替えのみであり，したがってシステムコールはスレッドの実行権の放棄である thread_reschedule() ただ一つである．プロセス間通信やスレッドの実行状態の遷移等は，カーネルの外にあるライブラリ関数で実現しており，これらの処理を高速に行うことができる．

システムサーバは，メモリやI/Oデバイス等のシステムのハードウェア資源の管理，タスクやスレッドの生成消滅，スレッドの実行状態の遷移の管理等を行う．サーバ間の独立性が高いので，システムサーバの改良や追加を容易に行うことができる．

2.2 タスクとスレッド

Thetaでは，並行処理の単位としてスレッドを用い，そのスレッドの管理とスレッドが共通して使用する資源を管理する単位として，タスクを用いている．プロセスに比べ，スレッドはコンテキスト切り替えに伴う退避させるべき状態の数を少なくすることが可能であり，いわゆる軽量プロセス(Lightweight Process)を実現できる．

タスク間通信やスレッドの実行状態の遷移等は，ライブラリ関数で実現している．また，割り込み禁止区間をコンテキスト切替え時のみにとどめることにより，割り込み応答性を良くしている．

2.3 タスク間通信

Thetaのタスク間通信は，メッセージパッシングに基づいている．メッセージの送信は，相手のタスクにではなくポートに対して行われる．受信においても同様にポートに対して操作を行う．スレッドの生成時に一つのポートが作られ，その後ユーザの要求により複数のポートを作ることができる．

非同期型のメッセージ送信は受信者がいなくても送信者は封鎖されず，送信されたメッセージはポートの中のメッセージキューにつながれる．

メッセージの受信は，ポートにメッセージが無いときには受信者のスレッドは封鎖される．この封鎖されたスレッドは，そのポートに他のスレッドがメッセージを送信する際に起こされ，メッセージを受け取って実行を再開する．メッセージパッシングにおいて同期をとる必要がある場合にはリモートプロシジャコールを行う．リモートプロシジャコールは，サーバとクライアント間のやりとりに使用されている．

2.4 相互排除と条件変数

Theta では，スレッド間の同期の機構として mutex[2] と条件変数 (condition variable) を用いている．

相互排除は，並行プログラミングにおける重要な問題の一つである．Theta では mutex を用いることにより相互排除を行う．mutex は locked と unlocked の二つの状態を持つ．ロックに成功した場合は直ちに復帰する．ロックに失敗した場合は，成功するまでロックを試みる．

条件変数は同期を取るための機構である．condition_wait() を実行すると，現在実行中のスレッドは自分をその条件変数のキューにつなぎ mutex をアンロックする．そして，そのスレッドは封鎖状態になり，他のスレッドが condition_signal() を実行するまで封鎖される．condition_signal() を実行すると，条件変数のキューを調べる．そして，キューに封鎖状態のスレッドがつながれていると，そのスレッドを実行可能状態にする．

その他，Theta では例外処理機能として，alert や signal も用意している．

3 オブジェクト指向の技法

オペレーティングシステムは，従来次のような手法で構築されてきた [4]．

(1) モノリシックアプローチ
(2) カーネルアプローチ
(3) オブジェクトアプローチ

(1)(2) は，いずれもシステムを階層化するアプローチであり，各機能や処理をどの層に置くのかを決定するのが問題となる．(3) のオブジェクトアプローチは，システムの機構及び操作を抽象化してオブジェクトとして表現し，クラス階層を利用することによってシステムを構築するアプローチである．

3.1 オブジェクト指向モデル

オブジェクト指向 OS といった場合，OS 自体をオブジェクト指向システムとして構築しているものと，オブジェクト指向アーキテクチャ内でサービスを提供するものとがある．さらに，OS をオブジェクト指向システムとして構築する場合，二つの方法が考えられる．それは，オブジェクトに基づく (object-based) システムとオブジェクト指向 (object-oriented) システムである [4]．オブジェクトに基づくシステムは，データだけでなくそのデータに対する操作も一つのオブジェクトとして定義している．これらの操作を定義することによりその動作が特性づけられ，外部に対する情報隠蔽が可能となる．このようにオブジェクトに基づくシステムは，定義と適用を実現から分離し，操作に基づいたインタフェースを提供している．また，オブジェクト指向システムでは，クラス階層を組み込んでおり，継承を使用している．この継承によりコードの共有·再利用が可能となりシステム全体に統一性のある記述が行える．

本研究では，オブジェクト指向システムに焦点をあてており，オブジェクトに基づくシステムとオブジェクト指向システムとを融合し，OS の記述に当った．

3.2 オブジェクト指向の利点

オブジェクト指向プログラミング言語は，抽象データ型とクラス，型の階層構造 (サブクラス)，継承などいくつかの大きな概念を軸に構成されている．そこでオブジェクト指向の技法を用いる利点として，次のようなことが挙げられる．

- 処理の対象となるデータ構造と操作をクラスを用いて抽象化できる．

- コードの共有·再利用が可能となる.
- システムの改良が容易である.
- マシン依存部分のカプセル化が可能である.

これらにより，システム全体に統一性のある記述ができ，その生産性·信頼性を向上させることができる．また，マルチメディア処理や分散処理などといった新しい分野への対応も容易になると考えられる．

3.3 Theta における抽象化

一般に OS の内部構造は極めて複雑であり，多種多様なデータを扱っている．そこで OS の処理の対象となるデータ構造と操作を抽象化し，システム全体に統一性のある記述を用いて，その生産性·記述性を向上させることを試みた．

3.3.1 データの抽象化

OS 内部において，データは，処理の対象となる一般的なデータであるオブジェクトと，このオブジェクトを管理するためのデータであるアレイとの二つに大きく分けられる．この一般的なデータであるオブジェクトを抽象化し，その抽象化したクラスを基本クラス base_object として定義した．

基本クラス base_object は，インスタンス変数にオブジェクトのアレイを構成するためのポインタである next，オブジェクトの型 type，オブジェクト識別子 id を持っている．また，メソッドとして，初期化，オブジェクト型の解放，型の取得のための関数 get_type()，型の設定のための関数 set_type() を持っている．ここで，初期化及び，オブジェクト型の解放の関数は，C++ では，コンストラクタである base_object() とデストラクタである ~base_object()，COB では，init() と final() となっている．

3.3.2 操作の抽象化

OS の内部の処理は，オブジェクトの管理と生成·消滅がそのほとんどを占めている．オブジェクトの生成は，新しいオブジェクトのためのメモリなどの資源の割り付け，オブジェクトの初期値の設定などを行う．オブジェクトの消滅により，オブジェクトが使用していた資源が解放される．その他，オブジェクトの管理としてオブジェクトのデータ構造への追加や取り出しなどがある．ここで，これら操作の抽象化に基づきアレイの基本クラスとして object_array を定義した．

アレイの基本クラス object_array は，インスタンス変数としてアレイの先頭と末尾のオブジェクトへのポインタ head と tail を持っている．そして，メソッドには，以下に示した様々なアレイの操作を行うものがある．

get()	指定したオブジェクトの取得
get_head()	先頭からオブジェクトを取得
put_tail()	末尾へのオブジェクトの追加
put_head()	先頭へのオブジェクトの追加
is_empty()	アレイの状態のチェック

3.4 Theta のクラス階層

前述したように，OS の内部の処理は，オブジェクト (メッセージ，スレッドなど) の生成 · 消滅と管理 (キューやリストの操作) がほとんどを占めている．そのため，基本クラスから各種のデータ構造に対応したクラスを継承することによって操作を統一することができ，プログラムの記述性·信頼性は格段に向上する．さらに，新しいクラスの追加が既存のクラスとの差分を記述するだけで容易に行える．

以下，3.4.1 節では，インスタンス変数，メソッドを含んだクラス全体の継承について述べ，3.4.2 節では，各クラスでの操作の意味を明確にするために，関数名を変えて継承しているものについて述べる．

3.4.1 クラスの継承

Theta におけるクラスは，基本クラスである base_object から object_array，その他各種のメ

ソッドを継承する(図1参照)．なお Theta では約35のクラスが定義されている．

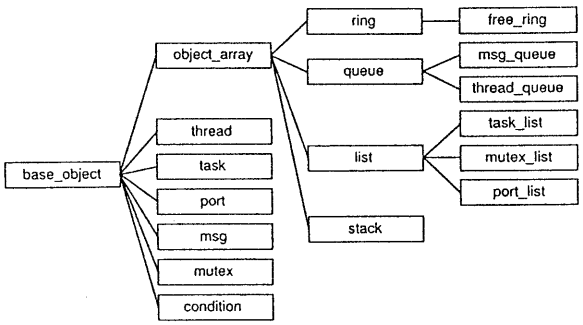

図 1: クラスの階層構造

object_array, thread, task, port, msg, mutex, condition などのクラスは，基本クラスである base_object のサブクラスとなっている．メソッドを継承するときには基本クラスである base_object のメソッドはそのまま受け継ぎ，それに各クラスで固有のメソッドを追加する．例えば condition クラスの場合は，条件変数のための管理情報や操作関数がメソッドとして追加されている．

3.4.2 操作の継承

Theta では，list, queue, ring, stack の四つのクラスは，object_array のサブクラスとなっている．object_array はオブジェクトが何らかの形で並んだ物であり操作には特に制限はないが，四つのサブクラスはそれぞれのオブジェクトの特徴を持っており，オブジェクトの並び方とその操作が異なる．list はオブジェクトの並び方がリスト構造になっており，オブジェクトの追加 (put) と，指定したオブジェクトの取り出し (get) がある．queue は FIFO(First In First Out) のデータ構造であり末尾への追加(enqueue) と先頭からの取り出し (dequeue) が定義されている．ring はオブジェクトがリング状に並んだものであり，基本操作に加えてリングを回転させる操作(rotate) を持っている．stack は LIFO(Last In First Out) のデータ構造で，末尾への追加 (push) と末尾からの取り出し(pop) が定義されている．

表 1に各操作の継承関係を示す．アレイの基本クラス object_array において定義されている操作を，継承したクラスで使用する場合は，object_array で記述されたプログラムをそのまま使っている．このとき各操作の意味を明確にするために名前を変えてメソッドを継承している．

表 1: 継承関係

class	object_array	ring	queue	list	stack
method	put_tail	put_tail	enqueue	put	push
	get_head	get_head	dequeue	—	—
	get	get	—	get	—
	—	rotate	—	—	—
	—	—	—	—	pop

4 各言語の比較

今回用いた言語のそれぞれの特徴について述べる．C++ は，C 言語にオブジェクト指向の機能を追加した言語であり，C のスーパーセットである．C++ は，クラス型を基本として，実行効率の高いオブジェクト指向プログラムの作成を可能にしている．COB もまた C 言語を基本としており，よりオブジェクト指向的で，かつプログラム開発効率の向上を特に重視したオブジェクト指向言語である．

4.1 インスタンス生成

C++ では private なメソッドはインタフェースに記述しなければならないが，COB では private なメソッドは一切インタフェースに現れない．そのためインスタンスの大きさを決定する際に，すべてのインスタンス変数が見えている C++ は高速にインスタンスにスタックを割り当てる（allocate）ことができる．一方，COB ではインタフェースにその情報がすべて現れるとは限らないので，インスタンスの大きさを決定することができず，スタックにインスタンスを割り当てることはできない．そのためインスタンスは特別な

コモン・メソッドにより，すべてヒープに生成される．

4.2 メソッドの動的結合

メソッドの動的結合に関する相違を述べる．C++ では，サブクラスで再定義され動的結合の対象となるメソッドは，スーパクラスで virtual というキーワードとともに宣言しなくてはいけない．動的結合は，メソッド・テーブルを通じての間接コールであるので，virtual という目印をつけることによって，コンパイラは virtual でない関数に対して高速なコードを出すことができる．これに対して COB では，すべての public なインスタンス・メソッドは C++ でいうところの virtual である．COB では，完成したプログラムに対する最適化を行うことによって，実行効率を向上させるというアプローチをとっている．最適化されたコードでは，実行時の様々なチェックはすべてはずされ，再定義されていないメソッドの呼び出しは直接コールに置き換えられる．チェックの解除は局所的な最適化だが，直接コールへの置換はそのプログラムが使用しているすべてのクラスの情報に基づく大域的な解析が要求される．

4.3 使用例

オブジェクト指向の機能である継承を使用した簡単な計算プログラムにより，その実行速度を調べた．C では継承の概念がないため，共用体を用いて式を表すデータ構造を作成した．このプログラムは，設定した値を読み込み，その内部表現(木構造)を構築し，この式の値を計算して結果を表示するものである．式は，整数，または式と式の加算，及び式を平方したものである．計測には，UNIX 上の clock() を用いた．$(1*1+2*2)$ と $(3*3+4*4)$ の計算を 10000 回繰り返す時に使用する CPU 時間を測定した．なおこの測定は，Sun3/60 上で行った．表 2にその結果を示す．

表 2: 実行速度

言語	C	C++	COB(最適化)
速度(秒)	5.47	8.84	8.93(8.09)
比率	1	1.623	1.631(1.479)

5 性能評価

オブジェクト指向機能を用いた場合には，ある程度のオーバヘッドを避けることはできない．ここでは，スレッドの生成と消滅に伴うオーバヘッドと，メッセージの送受信におけるオーバヘッド等の計測を行った．

5.1 オーバヘッドの軽減策

従来は，オブジェクト指向の技法は実行時のオーバーヘッドが大きく，OS などの速度が要求されるプログラムへの適用はあまりなされていなかった．特に，呼び出し先が動的に決定されるメソッドの呼び出しには，実行時の呼び出し先検索のオーバヘッドが伴う．Theta の C++ 版及び COB 版では，オブジェクト指向の技法を限定的に適用することによってオーバヘッドの軽減を図っている．以下に，その主な技法を示す．

- クラスの継承を最小限に抑える(最大 3 段)．
- 多重継承を使用しない．
- C++においては仮想関数を使用しない．

5.2 スレッドの生成と消滅

表3に示す，スレッドの生成と消滅におけるオーバヘッドは C++ で6.8%，COB では58.2%である．この実行時間は，スレッドの fork と join にかかる時間だけではなく，スケジューリング，コンテキストの切り替え，メッセージパッシングなどの全ての時間の総和である．但し，スレッドの pre-allocation や，デバッグ用のチェックを省けば実行時間はさらに短縮できる．

表 3: スレッドの生成と消滅

回数	C	C++	COB
100000	1149sec	1227sec	1818sec
1	11.49msec	12.27msec	18.18msec
比率	1	1.068	1.582

5.3 メッセージの送受信

メッセージ(非同期型)の送受信とリモートプロシジャコール(同期型)の処理時間を比較する．その測定は以下の方法で行った．

- 送り手(クライアント)から受け手(サーバ)へメッセージを送信する．
- 10 万回のメッセージの送受信を測定し，メッセージ 1 回当たりの時間を得る．
- メッセージ長が 20 バイトと 60 バイトの 2 種類を測定する．
- 時間の測定には Theta のシステムクロックを用いる．そのクロックサイクルは 1/61 sec である．

Theta のメッセージは固定長 20 バイトであるので，それ以上のメッセージを送るとすると何回かにわける必要がある．60 バイトであれば 3 回必要となる．表 4に評価結果をあげる．さらに，そのオーバヘッドをまとめたものが表5である．

表 4: メッセージの送受信速度

メッセージ		C (msec)	C++ (msec)	COB (msec)
非同期	20B	0.43	0.78	0.55
	60B	1.70	3.18	2.13
同期	20B	1.00	1.14	2.08
	60B	3.34	3.82	6.78

表 5: メッセージのオーバヘッド

	非同期		同期	
サイズ	20B	60B	20B	60B
C++	81.3%	87.1%	14.0%	14.4%
COB	28.2%	25.7%	109.2%	102.8%

5.4 コード量の比較

ここでは，Theta のシステム全体のサイズ及びそのソースコードの量を比較する．表6は，各言語を使用した場合のサイズと行数を示している．

表 6: コード量の比較

	C	C++	COB
オブジェクト	68K	56K	81K
ソース	8600 行	6100 行	6400 行

システムサイズに関してであるが，前にも述べた通り，オブジェクト指向を用いることにより，コードの共有·再利用が可能なため C++ においては，C 言語と比べ，サイズを抑えることができた．しかし COB では，オブジェクト指向であるにも関わらずシステムサイズは増えている．これは C++ では，コンパイラがネイティブなコードを生成するのに対し，COB では ソースファイルを C のソースにトランスレートし，さらにそれをプレリンクしてプレリンク·ファイルを作っているためである．すなわちこれら全てのファイルをリンクするために COB では，サイズが比較的大きくなる．そのため，COB では最適化を行うことによって実行効率を向上することができる．

ソースコードの量では，オブジェクト指向の利点が顕著に現れていることが確認でき，その生産性·記述性の向上が図れたことが分かる．

6 おわりに

Theta の設計を通して，その記述に用いた言語の特徴，及びオブジェクト指向の技法がどのように適用されているかについて述べた．従来の OS の開発に比べてその労力が少ないのは，用いた言語の高い記述能力にある．オブジェクト指向型言語である C++ や COB をシステムの記述全体に用いることは，実行効率を考えた場合，そのオーバヘッドが問題となる．しかしその反面，オブジェクト指向によってもたらされる記述の柔軟性によって，効率向上のためのシステムの改良が容易となると考えられる．この利点は，実行時オーバヘッドによる損失を十分補えるものである．

参考文献

[1] Accetta, M. et al.: Mach: A New Kernel Foundation for UNIX Development, Proc. of the Summer 1986 USENIX Conf. pp. 93-112 (1986).

[2] Birrell, A., Guttag, J., Horning, J. , and Levin, R.: Synchronization Primitives for Multiprocessor: Formal Specification, Proc. of the 11th Symposium on Operating Systems Principles, pp. 94-102 (1987).

[3] Birrell, A.: An Introduction to Programming with Threads, Research Report 35. Digital Equipment Corporation Systems Research Center (1989).

[4] Campbell, R., Johnston , G., Modany, P., and Russo, V.: Principles of Object-Oriented Operating System Design , University of Illinois, Report No . UIUCDCS-R-89-1510 , UILU-ENG-89-1729 (1989).

[5] Tamiya Onodera : COB Users Manual , IBM Research, Tokyo Research Laboratory(1990).

[6] 藤原正之，白川洋充，大野豊：分散オペレーティングシステム Theta のオブジェクト指向の技法を用いた記述，情報処理学会第 41 回 (平成 2 年後期) 全国大会.