オペレーティング・システム 56－9
(1992. 8. 19)

# オブジェクト指向オペレーティングシステム *Ozone*の プロセス管理部の実現

伊豆田 和也† 大久保 英嗣† 大野 豊† 白川 洋充††

† 立命館大学理工学部情報工学科

†† 近畿大学理工学部経営工学科

我々は，オブジェクト指向オペレーティングシステム *Ozone*の開発を進めている．*Ozone*プロジェクトの目標は，オブジェクト指向に基づくオペレーティングシステムの構成法を確立することである．*Ozone*におけるオブジェクト指向は次の２点に要約される．即ち，システム構成要素間の一様なメッセージの受渡しと，システムのクラス階層による構造化である．これらの２つの方針によって，アプリケーションプログラムのみならずシステム自体の移植性や保守性が大幅に向上する．*Ozone*のプロセス管理部は，すでに 80386 上に実装され各種評価を行っている．本論文では，Ozone プロジェクトの一環として行っている，Mach オペレーティングシステム上での実現について述べる．

# An Implementation of Process Manager in Object-Oriented Operating System *Ozone*

Kazuya Izuta† Eiji Okubo† Yutaka Ohno† Hiromitsu Shirakawa††

†Department of Computer Science and Systems Engineering,
Faculty of Science and Engineering, Ritsumeikan University
56-1 Tojiin Kita-machi, Kita-ku, Kyoto 603, Japan

††Department of Industrial Engineering,
Faculty of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka, 577, Japan

We are developing the object-oriented operating system *Ozone*. The major objective in *Ozone* project is to establish a construction method for the operating systems based on the object-orientation. The object orientation in *Ozone* is summarized into two major points: the uniform message passing between the system components and the class-hierarchical structuring of the system. By these two principles, not only application programs but also the system itself become more portable and maintainable than the conventional ones. The implementation on 80386 machine of the process manager of *Ozone* is already completed and various results of the evaluation are obtained. In this paper, the implementation of *Ozone* on Mach operating system as part of *Ozone* project is described.

## 1 はじめに

これまでのオペレーティングシステム (以下 OS と記す) の研究においては，OS の機能の多様化や性能の向上にのみ重点が置かれており，OS の構成法そのものが明確にされることはなかった．我々は，オブジェクト指向の概念に基づいた OS 設計 [1] がこの問題に対して解を与え，結果として OS 開発におけるコストの軽減や高い柔軟性を持ったシステムの実現に貢献すると考えている．

我々が規定しているオブジェクト指向の概念とは，一様なメッセージの受渡しによるシステム構成要素間の相互作用のモデル化と，システム構成要素のクラス階層における継承を利用したシステムの構造化の 2 点に要約される．これらの考え方に基づいて，我々はオブジェクト指向 OS *Ozone*の開発を進めている．*Ozone*プロジェクトでは，これまでに，相互作用を行うプロセスのモデル化の検討を行い，プロセス管理と同期及び通信機構の設計を終了している．さらに，プロトタイプシステムとして 80386 マシン上での実現と各種評価を行っている．

*Ozone*プロジェクトでは，OS の構成法へのオブジェクト指向の概念の適用の検討と並行して，各種マシン及び OS 上での実現も検討している．本稿では，その一環として，*Ozone*の Mach 上での実現法について述べる．即ち，Mach のタスク / スレッドモデルを用いて，これまでに実現した *Ozone*の機能をエミュレートする方法について述べる．Mach を仮想 OS と捉えることにより，ベアマシン上に直接 OS を実現することなく，設計した OS の各機能の仕様及び動作確認が可能となる．

以下，本論文では，*Ozone*の概要とプロセスモデルについて述べた後，Mach 上での実現法について説明する．

## 2 *Ozone*の概要

### 2.1 *Ozone*の構成要素

*Ozone*は，開発言語である Objective-C で提供されている Software-IC[2] と呼ばれる汎用クラスの他に，*Ozone*固有のクラスから構成されている．以下に，*Ozone*を構成する主要なオブジェクトと，その機能について述べる．(図 1参照)

(1) オブジェクトマネジャ

オブジェクトマネジャは，ObjectManager クラスのインスタンスであり，システムに存在するすべてのオブジェクトの集合体としてのオブジェクトである．*Ozone*内にオブジェクトマネジャは唯 1 つ存在する．*Ozone*上の全てのオブジェクトは，自身の生成時に自身をオブジェクトマネジャに追加し，自身の消滅時に自身をオブジェクトマネジャから削除する．

(2) プロセス

プロセスは，Proc クラスを祖先とするクラス群のインスタンスの総称であり，*Ozone* 上で CPU を利用して，ある処理を実行するオブジェクトである．OS 上で処理を行う実体には，タスクやスレッド，割込み処理など様々な形態のものが考えられる．*Ozone*では，これらを全てプロセスの一種であるとして統一的に扱っている．

(3) プロセスマネジャ

プロセスマネジャは，ProcessManager クラスのインスタンスであり，プロセスの集合体としてのオブジェクトである．*Ozone*内にプロセスマネジャは通常唯 1 つ存在する．プロセスマネジャは，スレッドを除く *Ozone*上の全ての実行可能なプロセスを保持し，そのスケジューリングを行う．

(4) スレッド

スレッドは，Thread クラスのインスタンスであり，Process クラスのインスタンス (以下 aProcess と記す) の中で 1 つの処理の流れを実行するプロセスである．スレッドは，aProcess が必要に応じて生成するものであり，aProcess の中でスケジューリングされる．

(5) スレッドマネジャ

スレッドマネジャは，ThreadManager クラスのインスタンスであり，aProcess の中に 1 個存在し，その中のスレッド群のスケジューリングの流れを規定するオブジェクトである．スレッドマネジャとスレッドは互いに協調してスレッドのスケジューリングを行う．

(6) ネームマネジャ

ネームマネジャは，NameManager クラスのインスタンスである．*Ozone*内にネームマネジャは唯1つ存在する．ネームマネジャは，aProcess の(システム内で一意の)名前と所在を管理するためのオブジェクトである．*Ozone*上の任意のプロセスは，ネームマネジャに対して名前をキーとしてaProcess の所在に関する問い合わせ行うことが可能である．

(7) メッセージ機構

メッセージ機構は，オブジェクト間の通信を可能とするものである．OS におけるメッセージには，通常のオブジェクト間のものの他に，割込みの形をとるハードウェアからのものが考えられる．*Ozone*では，前者はメッセージャによって処理され，後者は割込みメッセージャによって処理される．割込みメッセージャの機構により，ハードウェアからの割込みは，その割込みクラスに対応した処理が記述されているクラス (Interrupt クラスのサブクラス) に対するインスタンス生成のメッセージに変換される．

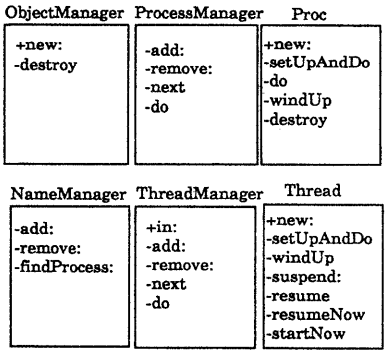



図 1　*Ozone*の構成オブジェクト

## 2.2　*Ozone*のクラス階層

*Ozone*を構成するオブジェクトの枠組として，種々のクラスが実現されており，それらは *Ozone*におけるクラス階層を構成している．*Ozone*のクラス階層は，図 2のようになっている．*Ozone*では，開発言語である Objective-C で提供されている汎用クラス (Software-IC) を利用してシステムを構成している．

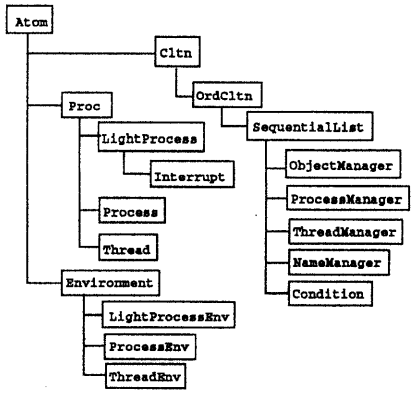



図 2　*Ozone*のクラス階層

Atom クラスは，*Ozone*上の全てのクラスの祖先にあたるクラスであり，スーパークラスを持たない唯一のクラスである，そのインスタンス生成メソッド (+new) および消滅メソッド (-destory) では，それぞれオブジェクトマネジャに対してメモリの確保およびメモリの解放のためのメソッドを呼び出している．これらのメソッドは *Ozone*のほぼ全てのクラスにおいて再利用されている．

また，各マネジャのクラスは，SequentialList クラスのサブクラスとして記述されており，その内部構造は，待ち行列となっている．Environment クラスは，プロセス実行のための環境とその操作を定義しているクラスである．Ozone における環境とは，プロセス生成時に生成される環境オブジェクトであり，実体はスタックである．

# 3　*Ozone*におけるプロセス管理方式 [3]

## 3.1　プロセスの枠組

*Ozone*のプロセスは，そのライフサイクルを基本に定義されている．ライフサイクルの各段階はメソッドに対応しており，そのそれぞれをユーザは

記述することができる．このため，従来の概念では，1つの関数として記述されていたプロセスが，*Ozone*ではメソッドの集まりとなる．

図2に示すように，プロセスの枠組であるProcクラスのサブクラスには，LightProcessクラス，Processクラス，Threadクラスの3つがある．

LightProcessクラスは単一の処理の流れ，Processクラスは複数の処理の流れよりなるプロセスのための枠組である．前者は後者に比べて，その生成，消滅およびコンテキストスイッチに要するコストが小さい．この両クラスの違いは，そのインスタンスがプロセス間のメッセージの受信を行うための機構(スレッド)を有しているか否かである．Processクラスのインスタンスはスレッドを持つが，LightProcessクラスはスレッドを持たない．そのため，LightProcessクラスには，スレッドマネジャは存在しない．LightProcessクラス及びThreadクラスは，メッセージの受信はできないが，送信は可能である．以上のプロセスモデルを図3に示す．

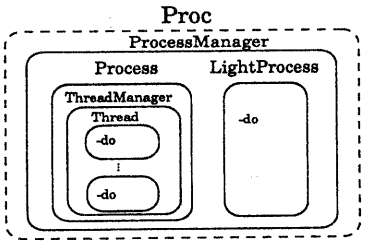



図3　*Ozone*のプロセスモデル

Procクラスは，以下のメソッドを持つ．

(1) プロセスの生成メソッド(+new:)
(2) プロセスの開始(再開)メソッド(-setUpAndDo)
(3) プロセスの実行メソッド(-do)
(4) プロセスの終了(中断)メソッド(-windUp)
(5) プロセスの消滅メソッド(-destroy)

+new:メソッドは新たにプロセスを生成しようとするプロセスが呼び出すメソッド，-destroyメソッドは自身または他のプロセスを消滅させるときに呼び出すメソッドである．-setUpAndDoおよび-windUpメソッドはプロセスマネジャが呼び出すメソッドであり，-doメソッドは-setUpAndDoメソッドにおいて呼び出されるメソッドである．

*Ozone*上で通常の応用プログラムを記述する場合，LightProcessまたはProcessクラスを選び，そのサブクラスを定義する．そして，プロセスのライフサイクルの1つの段階である“実行”に対応するメソッド(-do)のみを再定義し，そこにプログラムの内容を記せばよい．

### 3.2　プロセスマネジャ

プロセスマネジャは，実行可能状態にあるプロセスの集合を管理するとともに，プロセスの基本的な実行の流れを決定する．プロセスマネジャは，自身が待ち行列の形態をとっており，以下のメソッドを持つ．

(1)-add:　　・・・プロセスのマネジャ自身への挿入
(2)-remove:・・・プロセスのマネジャ自身からの削除
(3)-next　　・・・自身の先頭の要素の取り出し
(4)-do　　・・・プロセスのスケジューリングの実行

プロセスマネジャの中に存在するプロセス群はCPUを与えるべき順に並んでいる．新たにプロセスが追加された場合には，プロセスマネジャはそのプロセスを自身のどの位置に挿入するかを決定しなければならない．この決定には，各々のプロセスが持つ-compare:メソッドを用いる(3.5参照)．

### 3.3　スレッド

スレッドは，プロセスの中で一つの処理の流れを実行する実体であり，プロセスが受信したメッセージに対応するメソッドを実行するものである．スレッドもプロセスと同様，自身のライフサイクルの各段階に対応した以下のメソッドを持つ．

(1) 生成メソッド(+new:)
(2) 開始(再開)メソッド(-setUpAndDo)
(3) 終了（中断）メソッド(-windUp)
(4) 中断メソッド(-suspend:)
(5) 再開メソッド(-resume)
(6) 緊急再開メソッド(-resumeNow)

(7) 緊急実行メソッド(-startNow)

+new: は，スレッドの生成を行うメソッドである．生成されたスレッドは，スレッドマネジャに追加され，初期状態となる．-setUpAndDo メソッドが呼び出されるときのスレッドの状態は，初期状態と実行可能状態のいずれかであり，前者の場合には開始の処理を行い，後者の場合には再開の処理を行う．どちらの場合もその状態は，実行状態となる．-windUp は，スレッドの終了もしくは中断の処理を行う．-suspend: はスレッドを中断状態に移行させ，-resume はスレッドを実行可能状態にし，スレッドマネジャに加える．-resumeNow は，緊急再開フラグをセットし，再開メソッドを呼び出す．-startNow は，実行可能状態にあるスレッドをスレッドマネジャから削除し，緊急再開フラグをセットした後，再びスレッドマネジャに挿入する．

## 3.4 スレッドマネジャ

あるプロセスからメッセージを受信したプロセスは，対応するメソッドを実行するためにスレッドを生成する．*Ozone*におけるスレッドは，Process クラスのインスタンスである aProcess の中でスケジューリングされる．

Process クラスが自身のインスタンス aProcess を生成する際に，スレッドマネジャがそのインスタンスの中に 1 個生成される．プロセスマネジャが aProcess に対して，-setUpAndDo のメッセージを初めて送信した時，aProcess は自身の開始の処理を行うが，そこで自身の中にあるスレッドマネジャの -do メソッドを呼び出す．-do メソッドは，スレッドのスケジューリングを行っている．自身にスレッドがなくなると，スレッドマネジャは，自身が属している aProcess に対して中断のメッセージ (-suspend:) を送信し，aProcess をアイドル状態にする．

## 3.5 スケジューリング

### 3.5.1 プロセスのスケジューリング

新しく生成されたプロセスがプロセスマネジャに加えられると，プロセスマネジャは，実行可能状態にあるプロセスの待ち行列を，各々のプロセスが持つメソッド - compare: を基に構成する．

-compare: メッセージを受信したプロセスは，引数で与えられたプロセスと自身とを何らかの基準で比較して，“より大きい”，“等しい”，“より小さい” のいずれかの結果を返す．何を比較の基準とするかは，自身で定義されている - compare: メソッドの実装により異なる．Proc クラスの -compare: では，優先度の値を基準として用いており，Thread クラス以外のサブクラスではその -compare: メソッドが継承されている．従って，現在の *Ozone*では，プロセスマネジャ内のプロセス群に対しては優先度に基づくスケジューリングを行っている．

-compare: は，プロセス間の順序付けのためのキーを抽象化して返すものである．従って，-compare: の実装を変えることにより，プロセス間の順序付けを Proc のサブクラス単位で変更することが可能である．ただしこの場合，サブクラス間での順序付けに関して，*Ozone*上のプロセス全体を考えた場合のスケジューリングに矛盾が生じないように慎重な設定が必要となる．

### 3.5.2 スレッドのスケジューリング

スレッドマネジャ内のスレッドの順序付けも，プロセスマネジャの場合と同様である．しかし，スレッドマネジャは，挿入・削除のメソッドにおいて，自身が属している aProcess に対して優先度の再計算を行うメソッドを呼び出す場合がある．

aProcess の優先度は，自身が持つ実行可能なスレッドの優先度の内の最も高いものと定義される．そして，aProcess のクオンタムは自身と同じ優先度を持つスレッド群のクオンタムの合計とする．従って，スレッドマネジャに対してあるスレッドが追加され，その優先度がスレッドの属する aProcess のものと等しいかまたは高い優先度を持つ場合，スレッドマネジャは自身が存する aProcess に対し，優先度の再計算のメッセージを送信する．aProcess は，スレッドマネジャ内の各々のスレッドに対して優先度およびクオンタムの値を調べるメッセージを送信して，自身の優先度およびクオンタムを再計算する．

## 4 Mach上での実現

本章では，*Ozone*の機能をMach上に実現する方法について述べる．*Ozone*をMach上で動作させるために，先ず*Ozone*自体(*Ozone*のメインルーチン)をMach上の1つのタスクとして動作させ，さらに，*Ozone*を構成する各クラスを以下の方針で構成することとした．

(1) 各マネジャをMach上のタスクとして実現し，各マネジャ内のメソッドをMachのスレッドに対応させる．従って，各マネジャの+new:メソッドにおいてマネジャに対するMachタスクが生成される．さらに，-init:メソッドにおいて，当該クラスを構成するメソッドに対するスレッドが生成される．
(2) プロセス間の相互作用をシミュレートするために，Processクラス，LightProcessクラス，Threadクラスも Mach上のタスク/スレッドを用いて実現する．さらに，これらのクラス内のメソッドをMachのタスク/スレッドプリミティブを用いてシミュレートする．
(3) 上記以外のクラスに関しては，何も変更を加えない．
(4) 割込み処理は，Machの例外処理機構[4]を使用して実現する．

### 4.1 *Ozone*プロセスとMachのタスク/スレッド

*Ozone*プロセスの枠組をMach上に実現するために，*Ozone*プロセスとMachのタスク/スレッドを図4に示すように対応付けている．

#### 4.1.1 LightProcessクラス

LightProcessクラスは，-doメソッドからなる単一の処理の流れを表現するものであるから，Mach上では-doメソッドに対応する1個のスレッドを持つタスクとして実現される．そのライフサイクルは以下のようになる．

(1) LightProcessクラスの+new:メソッドが呼ばれるとLightProcessに対応するMachタスクが生成され，さらに，-doメソッドに対応するMachスレッドが生成される．その後，プロセスマネジャに当該タスクが登録される．
(2) プロセスマネジャによりスケジュールされたタスクは，-doメソッドに対応するMachスレッドを実行する．
(3) スレッドが終了するとタスクを消滅させる．

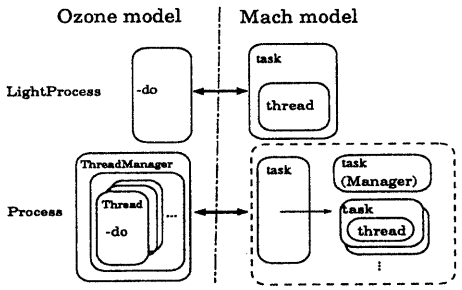



図4 *Ozone*とMachとの対応

#### 4.1.2 Processクラス

Processクラスは，複数の処理の流れを実現するために，各々の処理に対してThreadクラスを割り付けている．従って，Processは，Mach上では複数のタスクから構成される．そのライフサイクルは以下のようになる．

(1) Processクラスの+new:によりProcessに対応するMachタスク(親タスク)が生成される．さらに，*Ozone*スレッドを管理するためのスレッドマネジャを子タスクとして生成する．
(2) Threadに対応する子タスクを生成する．
(3) -doメソッドに対応するMachスレッドを生成する．
(4) 親タスクをプロセスマネジャに登録する．
(5) プロセスマネジャによりスケジュールされたタスクは，スレッドマネジャにより子タスクのスケジューリングを行う．
(6) 子タスク内で，-doメソッドに対応するMachスレッドを実行する．
(7) スレッドが終了するとタスクをアイドル状態にする．

### 4.2 メッセージ機構

*Ozone*におけるメッセージ機構は，メッセージャと呼ばれ，開発言語であるObjective-Cで提供されているものを使用している．オブジェクトに対してメッセージを送る場合は，[receiver selector]; の形のメッセージ式を使用する．メッセージ式において動作を行うオブジェクトをレシーバ，行うべき動作をセレクタと呼ぶ．

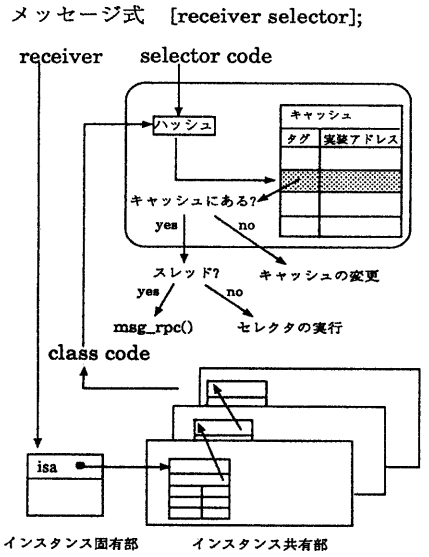



図5 メッセージャの機能

レシーバの共有部のアドレスは，receiver->isaをたどることで得られる．共有部にあるクラスコードとセレクタコードにより，セレクタの実装アドレスが求まる．メッセージャは，メソッドを実行時に検索するが，検索効率を上げるためにキャッシュを用いている．このキャッシュの各スロットには(クラス，セレクタ)のペアをコード化したタグとその実装のアドレスが設定される．タグが現在の(クラス，セレクタ)のペアと一致すれば，スロットの実現部に直接分岐し，一致しなければキャッシュ・スロットが変更される(図5参照)．

*Ozone*を構成する各マネジャ及びプロセスのメソッドをMachのタスク/スレッドとして実現すると，従来のメソッド呼び出しをMachのスレッド間の通信として実現しなければならない．このために，図6に示すMachインタフェース・スタブと呼ばれる*Ozone*とMachのインタフェースをとるスレッドを用意する．

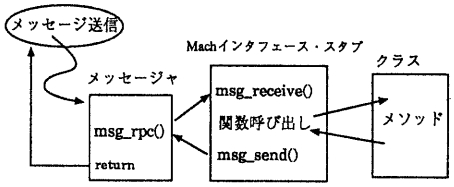



図6 Machインタフェース・スタブ

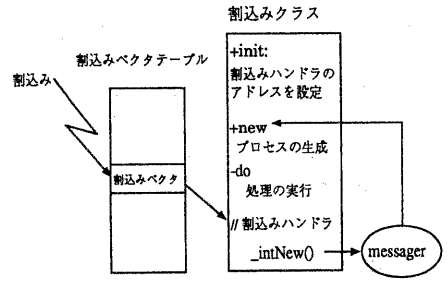
メッセージが送信されるとメッセージャにより送り先を確認し，相手がスレッドならば，メッセージャからMachプリミティブを用いて，Machインタフェース・スタブに制御が移る．スタブでは，実行されるスレッドを呼び出し，その結果をメッセージャに返す．メッセージャは，その結果をメッセージ送信元に返す．

### 4.3 割込み処理

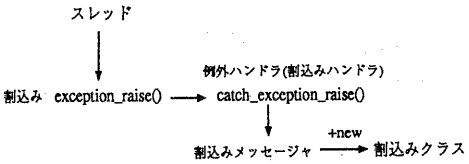
割込みは，ハードウェアからオブジェクトへのメッセージであると捉えることができる．従って，オブジェクトが対応したメソッドを実行できるように，割込みをオブジェクトが理解できるメッセージに変換する機構が必要となる．この機構を通常のオブジェクト間の通信に用いられるものと区別して，割込みメッセージャと呼んでいる．

*Ozone*における割込みは，ハードウェアからのオブジェクト生成メッセージであるとみなされる．従って，割込み処理は一種のプロセスであり，ライフサイクルに対応したメソッドを持ち，プロセスマネジャによってスケジュールされる．

Ozoneでは，割込みプロセス用のクラス(割込みクラス)はInterruptクラスとして定義されている．さらに，各々の割込みに対応した処理は，その




(a) Ozoneの割込み




(b) Mach上での割込み

図 7　割込みの制御の流れ

サブクラスとして定義される．図 7(a) に *Ozone*の割込み処理の流れを示す．先ず，割込みメッセージャ_intNew() により，以下の処理が行われる．

- 割込まれたプロセスの環境をスタックに退避する．
- 引数で与えられた割込みクラスに対して +new: メッセージを送信する．

次に，割込みプロセスが生成され，プロセスマネジャに登録される．そして，通常のプロセスと同じようにスケジューリングされ，CPU の割り当てを受ける．

Mach 上の実現では，ハードウェア割込みをシグナルを用いたソフトウェア割込みとしてシミュレートする．そのために，Mach の例外処理機構を用いる．exception_raise() によりスレッドに割込みがかかると，例外ハンドラである catch_exception_raise() に制御が移行する．このハンドラ内で，割込みメッセージャを呼び出し，環境の退避を行い，割込みクラスのタスク生成のメッセージが送られる (図 7(b) 参照)．

## 5　おわりに

本論文では，オブジェクト指向OS *Ozone*のプロセス管理部の Mach 上での実現について述べた．Ozone を Mach 上でエミュレートするために，Ozone の各マネジャとプロセスの概念を Mach のタスク / スレッドを用いてモデル化した．また，ハードウェア割込みは，Mach の例外処理機構を用いることで，ソフトウェア割込みとして実現した．今後は，プロセス管理部のみならず，プロセス間の同期及び通信機能を Mach 上に実現し，様々な評価を行っていきたいと考えている．

## 参考文献

[1] R. H. Campbell, G. M. Johnston, P. W. Madany, and V. F. Russo: *Principles of Object-Oriented Operating System Design*, Department of Computer Science, University of Illinois at Urabana-Champaign, Report No. UIUCDCS-R-89-1510, UILU-ENG-89-1729 (1989).

[2] B. J. Cox: *Object-Oriented Programming*, An Evolutionary Approach, Addison-Wesley (1986).

[3] 市岡 秀俊，安東 一真，大久保 英嗣，津田 孝夫: オブジェクト指向オペレーティングシステム Ozone におけるプロセス管理方式，情報処理学会論文誌，Vol. 32，No. 11，pp. 1401-1411 (1991).

[4] D. L. Black，D. B. Golub，K. Hauth，A. Tevanian，and R. Sanzi: *The Mach Exception Handling Facility*，CMU-CS-88-129 (1988).