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†

市岡秀俊††* 安東一真††
大久保英嗣††† 津田孝夫††

われわれは，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．これは，従来のオペレーティングシステムにおいては，モジュール分割の基準がなく，その構造モデルが明確でないことによる．Ozone におけるオブジェクト指向は次の点に要約される．すなわち，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．一様なそして統一されたインタフェースを使用することによって，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．また，システム構成要素をクラス化し，継承を利用することによって，再利用可能なソフトウェアが自然に推進されることになる．さらに，動的結合により，構成要素内のアルゴリズム（メソッド）の動的な置き換え（あるいは選択）が可能となる．現在，Ozone のプロトタイプシステムが完成している．本論文では，Ozone のプロトタイピングで得られた知見について述べる．さらに Ozone のプロセス管理について詳述する．

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 の構成法を明確にしないまま，各構成要素の機能や性能のみに重点が置かれていた．したがって，OS の構造モデルはいまだ明確でなく，また，客観的なモジュール分割の基準もないと言える[1),2)]．われわれは，オブジェクト指向の概念に基づいた OS 設計がこの問題に対して一つの解を与え，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．われわれが規定しているオブジェクト指向の概念とは，以下の2点に要約される．

（1） 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
（2） 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．

システムの構成要素間の相互作用のモデル化においては，ハードウェアとシステム間のインタフェース，システムとアプリケーション間のインタフェース，アプリケーション同士のインタフェースなど，従来の OS において個別に取り扱われていたものを一様なメッセージ通信の枠組のみでモデル化する．一様なそして統一されたインタフェースを使用することによって，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を大幅に向上させることが可能となる．また，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，コードの再利用が自然に推進されることになる．さらに，動的結合により，構成要素内のアルゴリズム（メソッド）の動的な置き換え（あるいは選択）も可能となる．

われわれは，以上の考えに基づいて，オブジェクト指向 OS Ozone の開発を進めている．ユーザの視点から Ozone と従来の OS とを比較すると，Ozone は以下の特徴を有している．

（1） ユーザがアプリケーションに応じて OS の機能を変更することが容易である．
（2） システム稼働時に，動的に OS の機能を変更することが可能である．
（3） システムプログラムおよびユーザ両者に対して，統一された理解しやすいインタフェースを提供している．

Ozone では，OS の構成要素間における共通の特質を持つ部分をクラスとしてまとめあげ，クラス階層における継承を利用してシステムを構成している．

---

† Process Management in Object-Oriented Operating System Ozone by HIDETOSHI ICHIOKA, KAZUMA ANDO (Department of Information Science, Faculty of Engineering, Kyoto University), EIJI OKUBO (Department of Computer Science and Systems Engineering, Faculty of Science and Engineering, Ritsumeikan University) and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科
††† 立命館大学理工学部情報工学科
* 現在 ソニー(株)総合研究所
Sony Corporation, Corporate Research Laboratories

Ozone のクラス階層においては，従来の OS の機能の多くが，アプリケーションプログラムと同じレベルで実現されている．これは OS およびアプリケーションプログラムの両者の処理において，共有できる概念が少なくないためである．これらの共通の概念は，OS にもアプリケーションプログラムにも利用することのできる部品となる．また，これらの部品を変更することにより，ユーザは各々の問題に応じて Ozone をカスタマイズすることが可能となる．

以下，本論文では開発を終了した Ozone のプロトタイプシステムの概要，設計時の検討項目およびプロセス管理方式について述べる．そして最後に，実験システムでの評価に基づいて Ozone のアプローチの有効性について検討する．

2. Ozone の概要

2.1 Ozone におけるオブジェクト

OS は，通常，CPU やメモリ，ファイルなどの資源オブジェクトの管理者の役割を担うが，Ozone はオブジェクトの管理者という立場はとらない．Ozone はオブジェクトの集合体である．集合内の個々のオブジェクトは，自分自身に関する情報をすべて持ち，抽象化されたメッセージに対して応答することのできる閉じた存在である．他のオブジェクトとのインタフェースの定義は内部の実装とは独立であり，実装が変更されても他に影響を及ぼすことはない．

実現レベルでは，Ozone を構成するオブジェクトはデータとコードを持つ．データは，多くの場合，資源に関する制御ブロック (control block) に対応する．また，コードは制御ブロックに対する操作に対応する．データは，オブジェクト内に隠蔽されており，そのアクセスは，原則的にすべてオブジェクトに対するメッセージによらねばならない．

Ozone の構成オブジェクトは二つに大別できる．一つは，論理的資源に一対一に対応したオブジェクトであり，もう一つは何らかの共通点を持つオブジェクトの集合としてのオブジェクトである．前者は，プロセス，主記憶，クロックなどに対応し，後者は，プロセス管理，記憶管理などの管理プログラムに対応する．従来の OS における管理プログラムとは，通常それぞれの管理する資源に対して，あらゆる操作の権限を持つものであった．しかし，Ozone は，既に述べたように，オブジェクトを管理するものではなく，各オブジェクトの協力を得てサービスを行う OS である．Ozone においては，管理されるものに，従来の管理プログラムが持っていた機能の一部が与えられている．

Ozone のユーザは，上で述べた二種のオブジェクトのいずれをも変更することが可能である．前者の変更は，OS の全体的なアルゴリズムないしは機能の変更に対応し，後者は，比較的小さなそして局所的な機能の変更に対応する．

Ozone では，以上の OS 構成オブジェクトに加え，ユーザが定義するオブジェクトも Ozone の構成要素となる．ユーザが定義するオブジェクトは，Ozone の構成オブジェクトに対してメッセージを送ることによって各種のサービスを受けることができる．従来の OS では，これはシステムコールを介して行われていたが，Ozone では通常のメッセージ送信*を用いて行われる．

2.2 Ozone の構成要素

Ozone のプロトタイプシステムは，開発言語である Objective-C で提供されている Software-IC と呼ばれる汎用のクラスのほかに，Ozone 固有のクラスから構成されている．本節では，プロトタイプシステムの主な構成要素であるオブジェクトマネジャ，プロセス，プロセスマネジャおよびメッセージ機構の概要について述べる．

(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，その時点でシステムに存在するすべてのオブジェクトの集合としてのオブジェクトである．Ozone 上のすべ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．オブジェクトマネジャは，Ozone 上のすべてのオブジェクトの主記憶上の配置を管理している．将来的には，様々な主記憶管理方式や通信管理方式などの実現において，オブジェクト検索のためのデータベース機能を追加する予定である．

(2) プロセス

プロセスは，Process クラスのインスタンスであり，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．タスク，スレッドおよび割込みなどは一括して Process クラスに属すると定義している．そしてこれらの異なった形態を持ったプロセスは，主にユーザが定義することで実現される．Process

* ここでのメッセージ送信はオブジェクト指向におけるメッセージの授受，すなわちメソッド呼び出しのことであり，OS 機能の同期および通信を意味しない．

クラスおよびそのサブクラスであるInterruptクラスは，そのための枠組であり，ユーザは通常これらのサブクラスとしてプロセスを定義することになる．Processクラスは，プロセスの制御ブロックと，すべてのプロセスに共通して見られるライフサイクルに対応するメソッド，すなわち，プロセスの生成・起動(再開)・終了(中断)・消滅のためのメソッドから構成されている．また，優先度やクオンタムに関する操作などプロセス制御ブロックに対する操作もすべて，Processクラス中でメソッドとして定義されている．さらに，プロセスは，自身の環境をオブジェクトとして保持している．Ozoneにおいて，プロセスの環境とは，プロセスの実行環境である自身の状態とスタックを保持するオブジェクトとして定義されている．実行環境の内容に従って，上述のメソッドの処理結果は異なりうる．

(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，プロセスの集合としてのオブジェクトである．プロセスマネジャは，Ozone上のすべ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．プロセスマネジャは，プロセスのディスパッチを行うに当たり，プロセスに対して，実行を“開始せよ”，“終了せよ”といった抽象的なメッセージを送信する．メッセージを受け取ったプロセスは，自身の状態に応じて対応する処理を行う．プロセスマネジャがスケジューリングを行う際に必要となるプロセスの順序付けは，プロセス自身が持つメソッドを用いて行われる．このように，プロセスとプロセスマネジャとの相互作用は抽象的なインタフェースに基づいているため，互いに独立性が高い．したがって，プロセスの実装とは独立に，様々なスケジューリング方式を実現することが可能となる．さらに，動的にスケジューリング方式を変更することも可能となる．

(4) 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にするものである．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ほかに，割込みの形をとるハードウェアからのものがある．Ozoneでは前者はメッセージャ(messager)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．生成されたインスタンス(割込みプロセス)は，自身をプロセスマネジャに追加し，他のプロセスと同様にCPUのスケジューリングを受ける．割込みメッセージャによって，ハードウェアを含めたOzoneを構成するすべてのオブジェクト間の相互作用は，一様なメッセージ通信の枠組の中でモデル化される．メッセージャはObjective-Cが提供しているもの[14]をもとに，並行処理に対応するため，再入可能になるように変更したものを用いている．

現在実現されているプロトタイプシステムは，以上の構成要素からなっており，優先度に基づく時分割スケジューリングを行い，first fitアルゴリズムによりメモリの要求に応えるOSとして動作する．

3. 設計時の検討項目

3.1 OSにおけるオブジェクト指向

OS開発におけるオブジェクト指向(object orientation)に対する期待は大きく，古くはケーパビリティの問題に対する適用に始まり[3]，最近では，各オブジェクトが並列に動作し得ることや，そのメッセージという統一されたインタフェースが自然に分散環境に対応することから，分散OSを中心に研究が盛んである．われわれは，OSの構成においてオブジェクト指向の概念を適用し，開発コストの軽減に寄与するようなOSのモデルを提案することを目標としている．このことは，OSを再構成する場合のコストも軽減させることになり，高い柔軟性を持ったシステムにつながる．

OS内部の処理の大部分は，計算機の物理的および論理的な資源を表構造に抽象化したものに対して，検索，挿入，変更などのデータベース操作を加えるものであると捉えることができる．したがって，OS核を資源に関する表(control block)を統括して管理するDBMS(Data Base Management System)と捉え，OS核の処理を表に対する操作として実現し，表操作のルーチンを共用することが考えられる[4]．このアプローチは，OS核内の処理の逐次性がボトルネックとなる可能性があるが，コード量は大幅に軽減されるといった利点を持つ．このような考えに基づく実現は，資源に対応した表をオブジェクトと捉えていると考えられる．また，OSを構成するオブジェクトは，それぞれ独立性が高く，並列的である[1]．このことに着目して，オブジェクト指向の概念をOS開発の規範とし

て導入し，OS のすべての構成要素をオブジェクトとし，それらの間をメッセージの形で通信する統一された構造を持った OS のモデルも古くより実現されている[5)~8)]．また，OS にオブジェクト指向を取り入れた場合，ユーザモードとシステムモードの違いがなくなると言われている[9)]．

Ozone では，OS をオブジェクト指向の概念に基づいて記述することによって，より分かりやすく，変更が容易な OS を小さい開発コストで構築可能であることを検証することに目標を設定している．このため，以下の二つを基本的な設計時の方針とした．

（1） 情報隠蔽および統一されたインタフェースによって，各資源オブジェクトを仮想化する．
（2） OS のすべての構成要素に対して，共通概念を見つけ出し，クラス階層を構築していく．そして，継承による徹底的なコードの再利用を促進する．

クラス階層を用いることによって，OS 全体として共通の機能を共通のコードで実現することが可能となる．また，種々の選択肢の集合をクラス階層で実現することにより，再構成を容易にすることが可能となる．ただし，Ozone では，OS を従来の（例えば，Dijkstra の階層モデルの）枠組の中で，ただクラス階層を用いて実現するのではなく，従来の枠組にとらわれることなく，新しい OS の構築法を開発することに重点を置いている．これまでのオブジェクト指向 OS は，従来の OS の階層構造の枠組をそのまま残して，機能別にクラス階層を構築していると言える．すなわち，オブジェクト指向の概念には重点を置いておらず，副産物的な捉え方をしている．われわれは，この点が Ozone と従来のオブジェクト指向 OS との違いであると考えている．

3.2 開発言語 Objective-C

Ozone は，OS の構成要素が互いに類似した機能を有していることに着目し，それらを同一のクラスに属するオブジェクトと捉えることで開発コストを軽減し，拡張性を向上させることを目的として設計されている．そのため，Ozone の実現に際しては，設計したクラスを容易にコードとして実現できる機構が言語にあることが望ましい．また，開発コストの軽減は，事実上，コードの再利用に依るところが大きいため，クラス階層における継承機能によってコードを再利用するための機能が必要となる．

さらに，既に述べたように，Ozone は，問題に適応して機能を変更できることを特徴の一つとしている．そして，この機能の変更は，静的に行えるのみならず，動的にも行えることを目指している．このためには動的結合の機能が必須となる（動的な機能変更については次節で詳しく述べる）．

以上のように，われわれは継承と動的結合のための機能を重要と考え，Ozone の実現においては，開発言語として C 言語にオブジェクト指向的機能を付加した混合型プログラミング言語 Objective-C を採用した[12)]．Objective-C は以下に挙げる特徴を有している．

（1） C 言語にオブジェクト型とメッセージ式を追加している．
（2） C 言語のプリプロセッサである．
（3） 単一継承機能を提供している．
（4） メッセージは動的結合によって実体と結びつけられる．
（5） クラスがプログラムモジュールの単位となる．

Objective-C (Release 3.3.1) には，20 余りの汎用性の高いクラス (Software-IC) がソースプログラムの形で提供されており[14)]，多くはそのまま利用できるため大幅に開発コストを軽減することができる．Objective-C には，Software-IC のほかメッセージ機構も一つのライブラリとして言語処理系とは独立に実現されている．そこで，Ozone では，このメッセージ機構を改造したものを構成要素の一つとしている (2.2 節参照)．

3.3 動的機能変更について

スケジューリングアルゴリズムなどの OS の機能を変更する方法として，オブジェクト指向の下で次の二つの実現法が考えられる．

（1） 異なる名前を持ったメソッドを同一クラス内に複数用意する．
（2） 同じ名前を持ったメソッドを異なるクラス内に実現する．

両者には，OS 機能をメソッドに対応させるか，クラスに対応させるかの相違がある．また，前者はメッセージのレベルで実装を指定しているが，後者はメッセージレベルでは実装が隠蔽されている．Objective-C は動的結合をサポートしているので，後者の方法によれば，メッセージのレシーバを変えることにより，動的に機能を変更することが可能となる．Ozone においては，OS のサービスを行うオブジェクトはオブジェ

クト型の大域変数として存在し，サービスの要求はこの大域変数に対してメッセージを送信することによって行われる．例えば，オブジェクト管理，主記憶管理，プロセススケジューリングを行うオブジェクトは，それぞれ TheObjMgr, TheMemMgr, TheProcMgr という変数に対応づけられている．このため，この大域変数の内容を変えることにより，サービスを行う実体を差し替えることが可能となる．図1にプロセススケジューリングアルゴリズムを変更する場合の例を示す．

```
        ⋮
/* ラウンドロビンによるスケジューリング */
        ⋮
    TheProcMgr=[MLFB new];
        ⋮
/* 多重レベルのフィードバックキューによるスケジューリング */
        ⋮
```

図 1 プロセススケジューラの動的変更
Fig. 1 Dynamic change of the process scheduler.

4. Ozone におけるプロセス管理方式

従来の OS は，それぞれに固有のプロセスの形態を持っており，それは OS の特徴でもあった．そして，プロセススケジューラなど OS 核の実現とプロセスの形態とは互いに大きく依存していたため，どちらかを変更するということはシステム全体を変更するに等しいことであった．Ozone では，両者間の相互作用を制御するものとして，抽象的なインタフェースが定義されており，互いの独立性が高い．したがって，各々の実現および変更が容易となっている．

4.1 プロセス

Ozone におけるプロセスは，CPU を争奪するオブジェクトの中で，プロセスマネジャによるスケジューリングの対象となるものである．このように定義したプロセスには種々の形態のものが考えられる．例えば，UNIX の子プロセスは親プロセスの複写を取ることによって生成されるものである[15]．Mach におけるスレッドは，スタックのみ独自のものを持つものであり，同一のタスクに属するスレッド間のスイッチはスタックの切り替えだけで済むようになっている[16]．さらに，割込み処理も一種のプロセスと考えられる．Mach が UNIX のプロセスをタスクとスレッドに分割したのとは対照的に，Ozone では，プロセスの定義を広げることによって，形態の多様化に対処している．

上述のような様々な形態のプロセス群を抽象化して得られる共通の特性は，そのライフサイクルが生成，起動，再開，中断，終了，消滅の六つの段階に分けられることである．そして，プロセスの形態の違いは，各段階における処理内容の違いであると考えられる．

```
= Process : Atom
{   id env;                                     // インスタンス変数
    int priority;
}
+new:(unsigned)aPriority with:(unsigned)aStackSize {
    self = [super new];                         // プロセス本体の生成
    priority = aPriority;                       // 優先度の設定
    env = [Env new:aStackSize];                 // 環境の生成
    [TheProcMgr add:self];                      // プロセスマネジャへ挿入
    return self;
}
-(void)setUp {
    if ([env status] && SUSPENDED) {
        [env recover];                          // 中断されていた時の再開
    } else {
        [env set];                              // 最初の起動
    }
}
-(void)do {
    [self subclassResponsibility:_cmd];         // サブクラスで記述
}
-windUp {
    if ([env status] && SUSPENDED) {            // 割込み処理
        switch ([env status]) {
        case TICK:                              // タイマ割込みに対する処理
            [TheProcMgr remove:self];           // ラウンドロビン
            [TheProcMgr add:self];
            break;
            ⋮
        }
    } else                                      // プロセスの正常終了時の処理
        [self destroy];                         // 自身の消滅
}
-destroy {
    [TheProcMgr remove:self];                   // プロセスマネジャから削除
    [env destroy];                              // 環境の消滅
    [super destroy];                            // プロセス本体の消滅
    return nil;
}
-(int)compare:anotherProcess {
    int anotherPriority;
    if ((anotherPriority = (int)[anotherProcess priority])
            == priority)
        return 0;
    return ((priority < anotherPriority) ? 1 : -1);
}
=:
```

図 2 Process クラスの定義
Fig. 2 Definition for the Process class.

Ozone のプロセスは，各段階の処理を実現するものとして，次の五つのメソッドを持つ（図2参照）．

（1） プロセスの生成メソッド（+new)*
（2） プロセスの初期化メソッド（−setUp)**
（3） プロセスの実行メソッド（−do)**

* Objective-C において，＋の接頭辞の付くメソッドはファクトリメソッドと呼ばれ，通常，クラスに対して呼び出されるメソッドである．そのクラスのインスタンスが生成される．
** 実現上は(2)(3)は一つのメソッド −setUpAndDo として実現されているが，説明の都合上分離している．

(4) プロセスの終了メソッド (−windUp)

(5) プロセスの消滅メソッド (−destroy)

これらのメソッドは，プロセスの静的な処理を行うものと動的な処理を行うものの二つに大別できる．すなわち，(1)および(5)は，プロセス生成元のプロセスが呼び出すメソッドであり，領域の確保および解放などを行う静的なものである．(2)，(3)，(4)は，生成されたプロセスに対してプロセスマネジャが呼び出すメソッドであり，実行系の各段階に対応した動的なものである．

以下，各メソッドの処理内容について詳述する．

(1)は，プロセスのための領域を確保し，その制御ブロックの初期化を行うメソッドである．このメソッド内で生成されたプロセスは自身をプロセスマネジャに加え，待ち状態になる．このメソッドに対応するものとして UNIX の fork()がある．

(2)は，CPU を与えられたプロセスが開始あるいは再開のための準備を行うメソッドである．主に環境の切り替えが行われる．Ozone において，環境とはプロセスの実行環境であるスタックとプロセスの状態を保持するオブジェクトである．図2ではクラス Env として実現されている．Mach におけるようなタスクとスレッドの違いは主にこの(2)の部分に現れる．

(3)は，プロセスの処理内容を記述したメソッドである．プロセスのメインルーチンである．通常の応用プログラマは，このメソッドのみを記述する．

(4)は，プロセスのライフサイクルにおける中断と終了とに対応している．プロセスは正常終了した場合および割込みにより処理が中断された場合に，このメソッドを実行する．このメソッドでは，ここに至った原因によって，その処理が異なる．すなわち，実行が正常に終了した場合は，当該プロセスの消滅メソッド(5)を呼び出す．割込みによって実行を中断された場合は，環境の退避を行う．そして，割込みの各要因に対応する処理を行う．また，例外に対する処理もここで行われる．

(5)は，プロセスが保持していた資源や自身の解放などを行うメソッドである．

プロセスの初期化や終了は，従来の OS ではシステム側の責任の下で行われてきた．そして，プロセスの形態は画一的なものしか許されなかった．これに比べ，Ozone では，Process クラス単位で異なった形態のプロセスを記述し，Ozone 上に実装することが可能である．これは，プロセスマネジャが種々のプロセスの違いを意識することなく記述されていることに起因している．

Ozone では，プロセスの枠組として図2に示す Process クラスを提供している．ユーザには自由度の高いプロセス記述能力が与えられている．通常，ユーザは Process クラスのサブクラスとして −do メソッドのみを再定義したものを記述する．このレベルでは，プロセス記述能力に関する限り従来の OS の場合と同様である．しかし，例えば，時間制約の厳しい実時間処理を行うプロセスなどを実現する際には，+new メソッドや −setUp メソッドを新たに記述することで対処できる．例外処理が重要な問題であれば，−windUp メソッドをそのために用いることが可能である．

4.2 割込み処理方式

本節では，Ozone における割込み処理の記述法およびその実現方式について述べる．

前節で述べたように，Ozone において，割込み処理は一種のプロセスである．したがって，割込み処理を行うプロセス（以下，割込みプロセスと呼ぶ）も前節で述べたようなライフサイクルに対応した五つのメソッドを持ち，プロセスマネジャによってスケジュールされる．通常のプロセスと異なるのは，その生成が割込みによって行われることである．Ozone においては，割込みはハードウェアからのプロセス生成のメッセージとみなされる．**図3**に割込みプロセス用のクラス（以下，割込みクラスと呼ぶ）の定義の例として，クロック割込みのものの概要を示す．図3の1行目に示すように，Timer クラスは Interrupt クラスのサブクラスである．Ozone ではこのように，各割込み処

```
= Timer : Interrupt
{}
+(void)init {
  割込みベクタテーブルに割込みハンドラのアドレスを設定し，
  割込みコントローラなどの初期化を行う．
}
+new {
  タイマ割込みの数をカウントし，一定の間隔で，時刻を更新
  するためのプロセスを生成する．
}
// -setUp および-windUp は，Process クラスより継承している．
// (図2参照)
-(void)do {
  時刻を更新し，表示する割込みプロセスの本体の記述．
}
static void intClockEntry() { /* 割込みハンドラ */
  タイマ割込みが発生するとこの関数が呼ばれ，割込みをメッ
  セージ式に変える．
        intNew(Timer);          // [Timer new];
}
=:
```

図3 Timer クラスの定義
Fig. 3 Definition for the Timer class.

理に対応した割込みクラスは Interrupt クラスのサブクラスとして定義される.

一般に，割込みクラスの定義中には，Cの関数である割込みハンドラが定義されている．図3では，int-ClockEntry が割込みハンドラとなっている．割込み処理を行うためには，割込みベクタテーブルにこの関数の番地を登録しておかなければならない.

割込み発生後の制御の流れを以下に述べる.

(1) 割込みが発生すると，割込みベクタテーブルを介して，割込みハンドラに制御が移行する．割込みハンドラの内容は，対応する割込みクラスに対して +new メッセージを送信するメッセージ式のみである．図3で _intNew(Timer) とは，2.2節で述べた割込みメッセージャを介して，Timer クラスに対して +new メッセージを送信するものである.

(2) 割込みメッセージャは，まず割り込まれたプロセスの環境をそのスタックに退避する．そして，引数として与えられた割込みクラスに対して，+new メッセージを送る．割込みメッセージャを介して呼び出されるメッセージは，割込みクラスの +new だけである.

(3) 割込みクラスの +new メソッドは，自身のインスタンスである割込みプロセスを生成し，そして，それをプロセスマネジャに対して登録するためにプロセスマネジャの −add: メソッドを呼び出す.

(4) プロセスマネジャに登録された割込みプロセスは，通常のプロセスと同様にスケジューリングされ，CPU の割り当てを受ける．通常，割込み処理の本体である −do メソッドは割込み可能状態で実行される．割込み処理のために他の割込みが禁止されるのは +new メソッドの実行中のみである.

例外的に，割込みによってはプロセスとして処理されないものもある．例えば，タイマ割込みや電源異常例外などである．このため，Ozone においては，時間に対する要求が厳しい処理や処理の続行が困難な割込みに対しては，割込みプロセスを生成せずに，そのための処理を +new メソッド中に直接記述することが可能となっている.

4.3 プロセスマネジャ

プロセスマネジャは，スケジューリング待ちの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．プロセスマネジャの枠組である ProcessManager クラスは，Objective-C の Software-IC である SequentialList をスーパクラスとし，−next および −do が追加され，−add: および −remove: が変更されたものである.

プロセスマネジャは，他のオブジェクトから見ると

(1) −add: …プロセスの待ち行列への挿入
(2) −remove: …プロセスの待ち行列からの削除
(3) −next …待ち行列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

の各メッセージに応えるオブジェクトとして見える．さらに，プロセスに対しては，−setUpAndDo (−setUp および −do)，−windUp，−compare: といった抽象度の高いメッセージを送信するものである．ProcessManager クラスの定義を図4に示す.

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 (add: メソッド)，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: を送られたプロセスは，引数で与えられたプロセスと自分自身とを何らかの基準で比較し，結果を “より大きい”，“等しい” あるいは “より小さい” の何れかとして返す．何を比較の基準とするかは，−compare: を送られたプロセスの属するクラス，すなわち，Process クラスのサブクラスの定義に基づくものであり，例えば，優先度，全サービス時間，メモリの使用量，デッドラインなどが考えられる．プロトタイプシステムの Process クラスでは，そのインスタンス変数の一つである優先度 priority を用いている (図2の −compare: メソッド参照)．すなわち，優先

```
= ProcessManager : SequentialList {}
+new {
    self = [super new];
    return (TheProcMgr = self);
}
-add:aProcess {
    // Process クラスの compare: メソッドを用いて，aProcess
    // を待ち行列に挿入
}
-remove:aProcess {
    // aProcess を待ち行列から削除
}
// next メソッド（待ち行列の先頭の要素を返す）は List クラス
// から継承する.
-do {                    // プロセススケジューリングの流れ
    while ((TheProc = [TheProcMgr next]) != nil) {
        [TheProc setUp];          // プロセスの初期化
        [TheProc do];             // プロセスの実行
        [TheProc windUp];         // プロセスの終了（中断）
    }
}
=:
```

図4 ProcessManager クラスの定義
Fig. 4 Definition for the ProcessManager class.

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．したがって，－compare：の実装を変えることにより，プロセス間の順序付けをProcessクラスのサブクラス単位で変更することが可能となる．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．

本節では，最後に，プロセスとプロセスマネジャの相互作用，特に環境の切り替えについて簡単に説明する．プロセスマネジャは，－nextメソッドによって実行可能状態にあるプロセスの待ち行列の先頭のプロセスを選択し，当該プロセスに対して －setUpAndDoメッセージを送信する．プロセスの －setUpAndDoメソッド（実装上の容易さのため，－setUpメソッドと －doメソッドを一つのメソッドとして実現したもの）は，オブジェクト間のメッセージによって呼び出されるものとしては，その終了の仕方が他のメソッドと異なっている．すなわち，当該プロセス自身の－setUpAndDoメソッド本体の処理に引き続いて終了する場合のほかに，ハードウェアからの割込みメッセージないしは中断メッセージがプロセスに送信された際にも終了する．－setUpAndDoメソッドが終了すると，環境がプロセスのものからプロセスマネジャのものに切り替わり，プロセスマネジャは自身の－doメソッドの処理を続行する．すなわち，プロセスマネジャは，次にカレントプロセスに対して －windUpメッセージを送信する（図4の －doメソッド参照）．以上のようにして，プロセスとプロセスマネジャ間の実行環境の切り替え処理が行われる．

4.4 スケジューリングアルゴリズム

一般的に任意のプロセススケジューリングアルゴリズムは次の三つを定義することで決定される[10),11)]．

（1） 優先度関数
（2） 調停規則
（3） 決定モード

優先度関数は，スケジューリングにおける順序を計算するものである．計算のパラメータとなるものはどのようなものでも良い．調停規則とは，優先度関数の評価値が同一の時にどのように順序付けを行うかを規定するものである．決定モードは，スケジューラにおける順序の計算を何時行うかを決定するものである．通常，決定が行われるのは，プロセスが待ち行列に加えられたとき時，閉塞した時，終了した時および割り込まれた時である．これにより，横取りの有無やクオンタム方式，時間依存方式などが決定づけられる．

Ozoneにおいて，上記の三つがそれぞれ何に対応づけられるかについて述べる．優先度関数はProcessクラスの －compare：メソッドに対応する．調停規則は，プロセスマネジャの実装（－doメソッド）による．例えば，プロセスマネジャに新たに加えられたプロセスに対して，既に待ち行列のメンバとなっているプロセスを引数として －compare：メソッドの呼び出しを行った時に，その戻り値が“等しい”であった場合を考える．両者の順位をいかにするかで，FIFOかLIFOかのいずれのスケジューリング方式に基づくかが決定される．しかし，調停規則は一般に優先度関数の中で解決することが可能である[10)]．例えば，FIFOやLIFOであれば，プロセスがスケジューラに加えられた時刻を優先度関数のパラメータとすれば良い．しかし，それは単に実装を複雑にするだけであり，調停規則を優先度関数から独立させた方が実現が容易である．したがって，Ozoneでは，プロセスマネジャの－doメソッドで調停規則を実現できるようにした．決定モードに関しては，Ozoneではプロセスマネジャに加えられた時のみ計算を行う．このため，UNIXのような時間依存型のスケジューリングアルゴリズムを実現する場合には，優先度を再計算するたびに，各々のプロセスを一旦スケジューラから取り除き，再び加える必要がある．例えば，ラウンドロビン方式の場合も，プロセスの －windUpメソッド内で，スケジューラに対して －remove：および －add：が送られる．

このように，スケジューリングアルゴリズムはOzoneのメソッドに対応づけられており，これらを変更することによりアルゴリズムを変更することが可能となっている．

5. 評　価

一般に，オブジェクト指向を用いて実現されたソフトウェアは，実行効率が問題となる．これは，主に情報隠蔽に基づくインタフェースの代償であると考えられる．また，メッセージに対応するメソッドを動的に探索する場合は，さらに，実行効率の低下を招く．Objective-Cでは，関数呼び出しと比較した場合，その約2.5倍のオーバヘッドがあると報告されている[12)]．そこで，Ozoneのプロセス管理に関して次の三つの項目に対する処理時間を測定した．すなわち，プロセス

の生成および消滅に要する時間，コンテキスト・スイッチのオーバヘッドおよび割込み処理のオーバヘッドである．測定は，日本電気(株)製の PC9801RA (CPU は 80386，クロックは 16 MHz) 上で行い，PC 9801 のシステムクロック（秒単位）を用いた．以下，順に測定結果について考察する．

(1) プロセスの生成および消滅

プロセスの生成・消滅の処理は，環境の生成・消滅，プロセスの待ち行列への挿入・削除，プロセスの領域の確保・解放からなる．これらの各々に関して，タイマ割込みを禁止した状態で一万回繰り返し実行させ，一回あたりの処理時間を算出した．測定結果を**表 1** に示す．表 1 より，プロセスの待ち行列への挿入・削除の処理時間は 1.8 ミリ秒となっており，他の処理時間に比較してほとんど無視できる時間となっている．しかし，環境の生成・消滅，プロセスの領域の確保・解放は，ほぼ同程度の時間の 38 ミリ秒となっており，非常に時間がかかっていることが分かる．これは，両者の処理において主記憶上の領域の動的な確保・解放処理を行っているためであると考えられる．

(2) コンテキスト・スイッチのオーバヘッド

ここで言うオーバヘッドとは，処理の本体を持たない（すなわち，空の −do メソッドしか持たない）プロセスの −setUp，−do および −windUp メソッドを順次実行するのに要する時間である．結果は 1.2 ミリ秒であった．これは，ユーザが記述するプロセス本体の処理に比較して問題とならない値であると考えられる．

(3) 割込み処理のオーバヘッド

割込みとして，タイマ割込みを用いた場合の測定結果は約 460 ミリ秒であった．測定結果に含まれるのは，割込み発生時に実行されていたプロセスの環境の退避，中断 (−windUp) および再開 (−setUp) の処理時間および割込みの割込みメッセージへの変換の処理時間である．すなわち，この割込みは割込みプロセスとして処理を行っていないものであり，プロセスとして行う場合にはさらに(1)のプロセス生成・消滅に必要な時間 (77.9−1.8=76.1 (ミリ秒)) が加算される．したがって，割込みの発生に対して高速な処理を行う必要がある場合は，各割込みに対応したクラスの +new メソッド中にその処理を直接記述しなければならない．その場合は，約 460 ミリ秒のオーバヘッドで処理を開始することができる．

以上の測定結果より，オブジェクトの生成および消滅に要する処理時間が問題となることが分かる．特に，これらの処理においては動的な空き領域管理を行っているため，この処理がボトルネックとなっている．その他の処理はほとんど問題にならないと考えられる．プロトタイプシステムでは，実現が容易であることから特別な空き領域管理を行っていない．したがって，実用的な空き領域管理を用いることによって，さらに効率の向上が期待できる．これに関しては，今後の課題である．

次に，プログラムの再利用に関して考察する．プロトタイプシステム構築のために新たに定義したクラスのメソッド数とステップ数を**表 2** に示す．プロセスマネジャやオブジェクトマネジャなどの OS の中枢をなすオブジェクトが非常に小さく実現されており，継承が有効に利用されていることが分かる．すなわち，Objective-C の Software-IC を用いることにより，開発コストが小さく抑えられていることが分かる．Ozone の開発には，単一継承を支援した Objective-C を用いたので，多重継承機能を利用することができなかった．しかし，Ozone は OS 構成オブジェクトの抽象化を基本的な設計方針としたため，単一継承によ

表 1 プロセス生成・消滅に伴う処理時間
(単位はミリ秒)
Table 1 Execution time of the process creation and deletion (Computed times are in msec).

処理内容	時間
環境の生成・消滅	38.1
プロセス待ち行列への挿入・削除	1.8
プロセスの領域の確保・解放	38.0
プロセスの生成・消滅	77.9

表 2 Ozone におけるクラスの統計
Table 2 List of class statistics in Ozone.

クラス	スーパクラス	メソッド数	ステップ数
Atom	—	61	804
Process	Atom	13	161
Interrupt	Process	5	46
Environment	Atom	11	140
Timer	Interrupt	8	179
ProcessManager	SequentialList	5	78
ObjectManager	SequentialList	8	245
SequentialList	OrderedCollection	3	57
Memory	Atom	8	117
Vectors	Memory	3	47
Window	Rectangle	22	610
FrameWindow	Window	12	123

る階層構造の形成で機能的に十分であると考えられる．これによって，OS の構造の単純化が促進されていると言えよう．これに対して，多重継承におけるスーパクラスはオブジェクトの抽象度を反映しているというより，部品としての性質が認められる[13]．多重継承を用いた場合，コードの再利用はさらに進むであろうが，OS の構造の明確さは損なわれると考えられる．

6. 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構成について述べた．さらに，本格的なシステムの開発に向けて，プロトタイプシステムの実行効率に関して性能評価を行った．その結果，プロセスの生成・消滅の際の領域の管理の処理効率が問題となることが分かった．本論文で提案した OS 構成モデルは，OS の開発コストを軽減し，さらに OS の可変性を実現するという当初の目的を果たしていると考えている．実際に，Ozone のプロトタイプシステムの実現は 0.5 人×年で完了している．提案したプロセスのモデルはメソッドの集合を形成しており，プロセスの形態もプログラミングの対象となるなど高いプロセス記述能力を持ち，一つの新しい OS の在り方を示していると言えよう．今後は，オブジェクトアクセスのケーパビリティチェック，プロセス間同期および通信と通常のメソッド呼び出しとの関連などについて検討し，さらに，Ozone 上でのアプリケーションプログラムの構築を通して，プログラミングスタイルへの影響に関して考察していきたいと考えている．

参考文献

1) 田胡和哉，益田隆司：オペレーティングシステムの構造記述に関する一試み，情報処理学会論文誌，Vol. 25, No. 4, pp. 524-534 (1984).
2) 野口健一郎：OS 記述論，情報処理学会オペレーティング・システム研究会報告，32-1 (1986).
3) 土居範久：オブジェクト指向とオペレーティングシステム，情報処理，Vol. 29, No. 4, pp. 359-367 (1988).
4) Turton, T.: The Management of Operating System State Data, *ACM Operating Systems Review*, Vol. 14, No. 2, pp. 21-24 (1980).
5) Cheriton, D.R.: *The Thoth System: Multi-Process Structuring and Portability*, Elsevier Science Publishers, N.Y. (1982).
6) Hansen, P.B.: The Nucleus of a Multiprogramming System, *CACM*, Vol. 13, No. 4, pp. 238-241, 250 (1970).
7) Baskett, F., Howard, J.H. and Montague, J.T.: Task Communication in DEMOS, *Proc. of the 6th Symposium on Operating System Principles*, pp. 23-31 (1977).
8) Rashid, R. and Roberston, G.: Accent: A Communication Oriented Network Operating System Kernel, *Proc. of the 8th Symposium on Operating Systems Principles*, pp. 64-75 (1981).
9) Jones, A.K.: The Object Model: A Conceptual Tool for Structuring Software, *Lecture Notes in Computer Science*, pp. 8-16, Springer-Verlag, New York (1978).
10) Leyens, D.E.: A Choices Implementation of the Universal Scheduling System, Technical Report No. TTR 89-15, Department of Computer Science, University of Illinois at Urbana-Champaign (1989).
11) Reschitzka, M. and Fabry, R.A.: A Unifying Approach to Scheduling, *CACM*, Vol. 20, No. 7, pp. 469-477 (1977).
12) Cox, B.J.: *Object-Oriented Programming—An Evolutionary Approach*, Addison-Wesley, Reading, Massachusetts (1986).
13) 梅村恭司，大里延康：Lisp 上のオブジェクト指向プログラミング，情報処理，Vol. 29, No. 4, pp. 303-309 (1988).
14) PPI Inc.: Objective-C Reference Manual for the PC Release 3.3.1 (1987).
15) Leffler, S.J., McKusick, M.K., Karels, M.J. and Quarterman, J.S.: *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Chapter 4, Addison-Wesley, Reading, Massachusetts (1989).
16) Accetta, M., Baron, R., Boloskey, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: A New Kernel Foundation for UNIX Development, *Proc. of Summer Usenix*, pp. 98-112 (1989).

(平成 2 年 9 月 10 日受付)
(平成 3 年 9 月 12 日採録)

市岡 秀俊（正会員）

1965年生．1988年京都大学工学部情報工学科卒業．1990年同大学院工学研究科修士課程修了．同年(株)ソニー入社．現在，ハイパーメディアシステムの研究開発に従事．オペレーティングシステム，マルチメディア，オブジェクト指向に興味を持つ．

安東 一真

1967年生．1989年京都大学工学部情報工学科卒業．1991年同大学院工学研究科修士課程修了．同年日経BP社(株)入社．オペレーティングシステム，並列処理，プログラミングパラダイムなどに興味を持つ．

大久保英嗣（正会員）

1951年生．1974年北海道大学理学部数学科卒業．1977年同大学工学部情報工学科大学院修士課程修了．同年(株)日立製作所ソフトウェア工場に入所．主としてFORTRANコンパイラの開発に従事．1979年より京都大学工学部情報工学科助手．1985年同講師，1987年同助教授，1991年立命館大学理工学部情報工学科教授，現在に至る．工学博士．オペレーティングシステム，データベースシステム等の研究に従事．日本ソフトウェア科学会，システム制御情報学会各会員．

津田 孝夫（正会員）

1957年3月，京都大学工学部電気工学科卒業．現在京都大学工学部情報工学科教授．計算機ソフトウェア講座担当．工学博士．自動ベクトル化/並列化コンパイラ，スーパコンピューティング，オペレーティングシステムの研究に従事．「モンテカルロ法とシミュレーション」（培風館），「現代オペレーティングシステムの基礎」（オーム社，共著），「数値処理プログラミング」（岩波書店）などの著書がある．昭和63年度情報処理学会論文賞受賞．本学会関西支部長．ACM, SIAM 各会員．IFIP/WC 2.5 (Numerical Software) 委員．