オペレーティング・システム　49－4
（１９９０．　１２．　７）

OS／omicron第３版の並列プログラミングモデル

早川栄一*，小松徹*，池尻宏*，
岡野裕之*†，横関隆*‡，並木美太郎*，高橋延匡*

東京農工大学　工学部　電子情報工学科
†現　日本IBM　　‡現　ソニー

本報告では，我々が研究，開発を行っているOS／omicron第３版（OMICRON V3）が提供するタスクとタスクフォースを用いたプログラミングモデルと，それらを含めたSVCセットについて述べる．OMICRON V3は，並列処理の実験環境をはじめとする各目的別OSを容易に生成可能にするため，
(1) タスクとタスクフォースを含めた計算機資源をオブジェクトという形での統一的管理
(2) オブジェクトへのアクセスメソッドを統一する
を行い，オブジェクトを設定するプリミティブとしてSVCを提供する．その結果，オブジェクトドライバと呼ぶ資源管理部を，OSの部品として扱うことが可能になった．

A Parallel Programming Model and a SVC set of OS／omicron Version 3

Eiichi HAYAKAWA,　Tohru KOMATSU,　Hiroshi IKEJIRI,　Hiroyuki OKANO,
Takashi YOKOZEKI,　Mitarou NAMIKI and Nobumasa TAKAHASHI

Department of Computer Science,　Faculty of Technology,
Tokyo University of Agriculture and Technology,
Koganei-shi,　184 Japan

This paper describes a parallel programming model and a supervisor call (SVC) set of OS／omicron version 3 (OMICRON V3). Main targets about this OS are:
(1) to provide an experimental environment for parallel execution.
(2) to ease construction of application oriented OS's based on OMICRON V3.
Important features of this OS are:
(1) computer resources, including tasks and taskforces, being unified as "object".
(2) a standard access methods for "objects" being unified.
As a result, resource manager named "object driver" is considered as a building module. A SVC set of OMICRON V3 is a primitive set to compose an OS from building modules.

1．はじめに

我々の研究室では，システムプログラミングをはじめとして，パターン認識，日本語情報処理，卓上電子出版（DTP），ヒューマンインタフェースなどの応用研究を手がけている．これらは，独自に開発したオペレーティングシステムOS／omicron上で研究，開発を行っている．

応用研究が進むにつれて，計算機の性能向上が求められている．このような要求は並列処理によって解決することが期待できる．そこで，マルチプロセッサによる並列処理システムの研究が重要になる．このため，我々は，各応用研究の目的に合った並列処理を行うことが可能なシステムを提供したいと考えた．

このような環境を構築するには，

(1) 単一のOSを用いて，各目的別の機能を追加していく

(2) 各目的別に専用のOSを生成して使用する

方法がある．(1) では，OSの複雑化，肥大化，それに伴う実行効率の低下を招いてしまう．(2) は，各応用研究をOSによって効率よくサポートすることが可能になるが，OS作成の手間がかかると実験環境としては不向きである．

このような問題を解決するためには，各目的別OSを構築するための共通の要素をカーネルとしてまとめ，目的別の要素は部品として選択することで，柔軟かつ効率のよいOSの実現が可能になると考えられる．

また，並列OSや応用研究の研究者が容易にOSを作成，実験するには，

・OSが提供する並列プログラミングモデルと資源管理の明確化

・OS自体の部品化

が可能になっていることが望ましい．

本報告では，並列処理を指向するタスクとタスクフォースを用いたプログラミングモデルについて述べ，それらを含めてOSの部品化を目的として構築したSVCセットについて述べる．

2．並列処理のための機能

我々は，並列処理の具体的対象として，手書き文字認識における辞書(データベース)の検索や，日本語の構文解析，フレームメモリへの並列描画系，囲碁の最良手決定などを目標にしている．これらは，データ空間を共有しながら実行するプロセッサ群という概念が有効である．この目的のために，まず，我々はタスクとタスクフォースというプログラミングモデルを導入する [1]．次にこれらについて述べる．

2．1 タスク

タスクは仮想プロセッサが割り当てられる単位である．タスクは，親子関係によるプログラミング上の制限をなくすために，フラットな構造である．

また，タスクには生成／終了の依存関係がない．生成したタスクが，生成元よりも先に終了してもよい．このようにすることで，多くのタスクを立ち上げた場合，終了待ちのSVCによる同期のオーバーヘッドを減らすことができる．

タスクの生成は，createを用いる．このとき，タスクとして立ち上げる関数のエントリを指定する．基本同期命令としてはセマフォを持っているが，セマフォもcreateによって生成する．

```
【タスクの生成】
タスクID   = ___svc_create(self_tf_id, TASK_TYPE, 0, &初期値);
【セマフォの生成】
セマフォID = ___svc_create(self_tf_id, SEMAPHORE_TYPE, 1, 0L);
```

リスト１　タスクとセマフォの生成

２．２ タスクフォース

タスクフォースは複数のタスクの集合体である．タスクフォースは，タスクと異なり親子関係を持つ．タスクフォースの生成は，実行形式ファイルのロードと生成で行われる．

```
ファイルID = ___svc_open(ファイルシステムID, "hd0ココマンドココピー", 読込みモード, 0);  …①
tf_id      = ___svc_create(self_tf_id, TASK_FORCE_TYPE, デバッグ, &初期化パケット);     …②
             ___svc_close(ファイルID);
mssgid     = ___svc_create(self_tf_id, MESSAGE_TYPE, tf_id, &メッセージ手続き);        …③
             ___svc_write(mssgid, TF_EXEC, NIL, NULL);                                …④
```

①実行形式ファイルのロード（つはパスの区切りである）
②タスクフォースとリーダタスクの生成
③タスクフォースのメッセージ経路の生成
④メッセージの送信

リスト２　タスクフォースの生成

タスクフォース内のタスクは，スタック以外のメモリ空間，セマフォ，ファイルを共有している．タスク間の通信は，共有データ領域を介し，セマフォを用いて行う．

タスクとタスクフォースの実行環境を図１に示す．

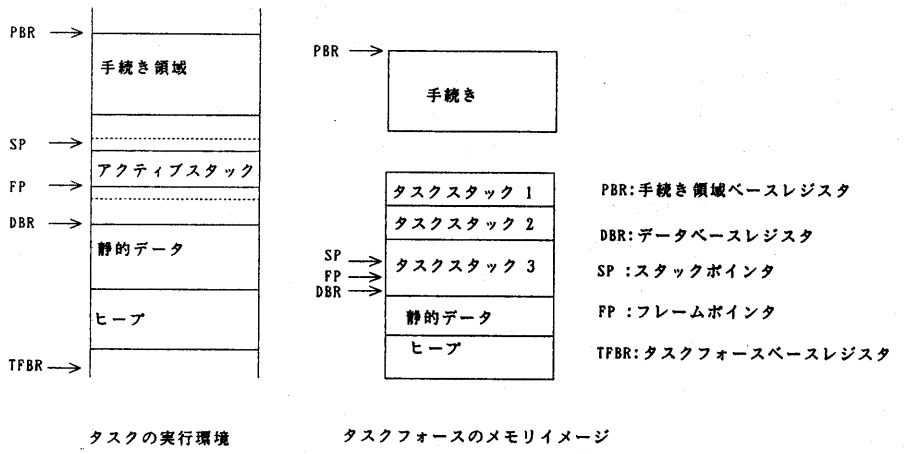

図１　タスクとタスクフォースの実行環境

また，生成以外のタスクとタスクフォースのSVCを次に示す．

このような環境を実現するために，我々は，共有型マルチプロセッサ向けのOSであるOS／omicron第３版（OMICRON V3）を開発した［2］．

次に，OMICRON V3が提供するSVCセットについて述べる．

表1　タスク，セマフォ関係ＳＶＣ

___svc_exit_task	タスクの終了
___svc_wait_task	タスクの終了を待つ
___svc_stop_task	タスクを停止する
___svc_kill_task	他のタスクを終了させる
___svc_my_task	自分のタスクIDを得る
___svc_get_priority	自タスクの優先度を得る
___svc_change_priority	自タスクの優先度を変更する
___svc_rchange_priority	自タスクの優先度を相対的に変更する

___svc_P_ope_sem	P命令を実行する
___svc_V_ope_sem	V命令を実行する

表2　タスクフォース関係ＳＶＣ

___svc_tf_exit	タスクフォースを終了する
___svc_tf_stop	TF内の全タスクを停止する
___svc_tf_stop_task	TF内のタスクを停止する
___svc_tf_kill	他のタスクフォースを終了させる
___svc_fd_arrange	タスクフォース間に通信路を結ぶ

3．OMICRON V3 SVCセットの設計方針

3．1　目的別OS生成のためのSVCプリミティブセットの設定

実験環境を構築するときには，環境の構築が迅速に行えることが重要である．そのためには，容易にOSを作成できる環境が必要となる．なぜなら，実際に目的別OSを作成するときに，一からOSを作るのでは手間が掛かりすぎるためである．そこで，OSの作成を補助するプリミティブを用意し，それらを用いて目的別OSを容易に作成したい．OSを構成するプリミティブが用意されていることは，OSの教育という観点からも効果があると考えられる．

3．2　資源の統一的管理

OSの役割の一つは，ハードウェアが提供するCPUや主記憶，ディスク，ビットマップディスプレイを管理し，タスクやウィンドウ，ファイルなどの形に仮想化することである．各資源が，それぞれの資源ごとに管理形態を変えると，それに対応するOSの管理部を作成しなければならず，OSの構造が複雑化する．例えば，OS／omicron第2版のSVCでは，ファイルとメッセージ通信で異なったSVCが用意されている．これは，管理形態が異なるためである．また，並列処理の実験環境を構築するためには，タスクで扱う資源を統一的に管理するほうがよい．これが異なると，タスクが異常終了したときに，使用している資源を解放する手続きをそれぞれ発行しなければならない．部品化を進めるためには，各資源管理部の管理形態は統一すべきである．

3．3　アクセスメソッドの標準化

ユーザプログラムから資源管理部へのアクセス形態が個々に異なると，ユーザプログラムとのインタフェースが増加する．OS／omicron第2版でも，デバイスへのアクセスはファイルと同じSVCインタフェースであるにもかかわらず，そのモデルに合わないビットマップディスプレイへのアクセスや仮名漢字変換システム，LBPのフレームメモリ操作系などはそれぞれ個別のSVCを持っている．このため，ユーザ側から見たとき，

(1) SVCインタフェースの数が増え，理解しにくい

(2) 資源管理部の作成の方針が決めにくい

という問題が発生した．ユーザが，自由かつ容易に資源管理部を追加するためには，

(1) 資源管理モデルを明確にする

(2) 資源管理部とユーザとのインタフェースを明確にする

ことが重要である．すなわち，インタフェースの規約化が重要であり，その部分を決めておくことで，**個々のデバイスに対する資源管理部はOSを構成する部品であるとみなすことができる．**

4．OMICRON V3のSVCセット

4．1 OSの構成

OSは，次の二つから構成される．

(1) OSを記述するためのプリミティブセットを提供するカーネル

(2) OSで仮想化する資源を管理するオブジェクトドライバ

(1) は，オブジェクトドライバを記述するためのプリミティブセットを提供するものである．オブジェクトドライバについては，４．７で述べる．

４．２ オブジェクトによる資源の統一化

普通のOSでは，各資源ごとに管理部を持ち，異なったアクセスメソッドでアクセスしている．３．２で述べたように，資源の統一的管理は重要である．OMICRON V3では，このような資源をオブジェクトと呼び，統一的に管理する．

オブジェクトは，ファイルのように実体を持ったものでもよいし，オブジェクトを管理するものも，またオブジェクトの一種である．

オブジェクトは，オブジェクトIDと呼ばれる番号が割り当てられる．オブジェクトへのアクセスは，このIDを通して行う．このオブジェクトIDは，タスクフォース内で一意である．このように，タスクフォースで閉じていることで，タスクフォースの凍結や，分散環境下でのタスクフォースの移動が容易になる．

４．３ メソッドによるアクセスの標準化

オブジェクトは，それを操作する手続きを含む．これをメソッドと呼ぶ．すべてのオブジェクトに対するインタフェースは，カーネルで用意されたオブジェクト操作SVCを用いてアクセスする．

表３　オブジェクト操作関係ＳＶＣ

___svc_create	オブジェクトからインスタンスを生成する
___svc_delete	インスタンスを削除する
___svc_open	インスタンスに対するアクセス権を得る
___svc_close	インスタンスに対するアクセス権を解放する
___svc_read	インスタンスからの入力を行う
___svc_write	インスタンスへの出力を行う
___svc_etc_ope	インスタンスへの操作を行う

オブジェクト操作SVCによって，資源をどのようにアクセスするかは，オブジェクトの性質によって異なる．例えば，ファイルに対するread／writeは，ファイル実体への入出力になるが，タスクフォースに対するread／writeは，タスクフォースが持っているメモリ空間へのアクセスになる．つまり，これらは各オブジェクトにとってメソッドであると考えることができる．

いずれの場合も，SVCが作用した結果は各オブジェクトによって異なるが，作用素としてはread／writeで統一化している．

４．４ 番地付けされたオブジェクト

並列に動作する二つのタスクから，一つのファイルをreadする場合を考える．このとき，ファイルの現在の読みだし位置を管理しているシステムでは，readするごとに読みだし位置が変わっていく．それを避けるには，

(1) 同じファイルについて二つのファイル記述子を得る

(2) 一つのファイル記述子を二つのタスクで共有し，相互排除を行う

しかし，このような方法では，タスクの数が増えた場合に，ファイル記述子を消費してしまう．そこで，オブジェクトに対して番地付けを行い，読み書き時にかならず番地を指定するようにする．このようにする事で，一つのファイル記述子を複数タスクで共有することができる．

また，このような番地付けは，ファイルに限らず，ウィンドウの領域や，フレームメモリなど

オブジェクト一般に適用できる．このように**オブジェクトに番地付けをすることで，オブジェクトに対する入出力は，すべて read／write という形で記述することが可能になる．**

4．5 性質の継承

オブジェクトをアクセスするときに，そのオブジェクトが持っている性質が継承される．次に，例を示す．

リスト３　性質の継承

```
obj_id1 = ___svc_open(-1, "fs", NULL, NULL);                          ···①
obj_id2 = ___svc_open(obj_id1, "fd0", NULL, NULL);                    ···②
obj_id3 = ___svc_open(obj_id2, "プログラム.c", 読みだしモード, NULL); ···③
```

(1) では，ファイルシステムオブジェクトドライバの使用権を得る．ファイルシステムはファイルの性質を持っている．read／write によって，ファイルシステムに関する情報が得られる．
(2) では，フロッピーディスク０の使用権を得る．obj_id2 は，ファイルシステムの性質とともに，フロッピーディスクの性質を受け継ぐ．read／write によって，フロッピー固有の情報が得られる．
(3) によって，プログラム．c というファイルの使用権を得る．プログラム．c へのアクセスは read／write を用いて行う．

4．6 アクセス権の継承

オブジェクト ID がタスクフォースで閉じていると，システム全体で共通の ID を持つことができない．そこで，親のオブジェクト ID が持つアクセス権を，子のオブジェクト ID に渡す機構を備えている．
頻繁に使用されるオブジェクト，例えば，ファイルオブジェクトなどは，この継承機構を用いて，インスタンスの検索の手間を減らすことが可能である．
継承関係の SVC を表４に示す．

表４　継承関係ＳＶＣ

___svc_fd_set_info	IDに引き継ぎ情報を設定する
___svc_fd_get_info	IDに引き継ぎ情報を得る
___svc_fd_bind	子TFにオブジェクトIDを引き継ぐ

4．7 オブジェクトドライバ

資源を管理し，オブジェクトという形で抽象化して提供する機能をオブジェクトドライバと呼ぶ．オブジェクトドライバ自身もオブジェクトになっている．
応用別 OS の生成は，オブジェクトドライバを作成，登録していくことで実現できる．オブジェクトドライバは，タスクフォースとして実現されるので，動的に登録することが可能である．
ユーザがオブジェクトを追加する場合は，次のことについて考えればよい．

(1) オブジェクト操作メソッドに対して，その資源をどのような操作に当てはめるのがよいか考える．
(2) (1) では記述できない機能については，___svc_etc_opeを用いて操作を行う．

ファイルオブジェクトについては，
create／delete → ファイルの生成／削除
open／close → ファイルのオープン／クローズ
read／write → ファイル入出力

また，すでに生成されているファイルに対してcreateを行うと，そのファイルに対してバージョン管理を作成することが可能になる．４．５のプログラムにおいて，

```
obj_id4 = ___svc_create(obj_id3, NEW_VERSION_TYPE, 読み書きモード, NIL);
          /*  obj_id3 はすでにオープンされたファイルのID */
```

を行うと，“プログラム．c”の新しい版を作成することができる．このような版管理の機能は，追記型光ディスクと組み合わせて，版管理や実行履歴の管理に用いることが可能になる．

また，ウィンドウオブジェクトに関しては，
create／delete → ウィンドウの生成／消滅
open／close → ウィンドウの各属性，例えば，ウィンドウ自身や，大きさや枠のありなし，ラベル名などへのアクセス権の獲得／解放
read／write → ウィンドウへの入出力

という形で実現できる．

また，open／closeの時に，ウィンドウの各属性を名前でアクセスできるようにする．このようにすることで，分散環境下で，異なったウィンドウへのアクセスが抽象化できる．

現在のシステムでのオブジェクトドライバの構造を図２に示す．太字は，オブジェクトドライバを示す．

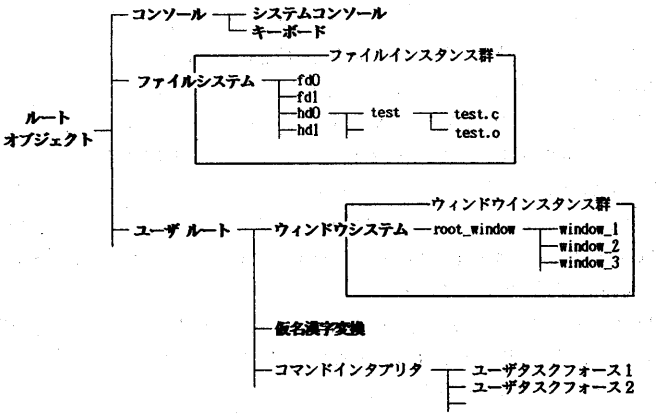

図２　オブジェクトドライバの階層

オブジェクトドライバ関係のSVCを表５に示す．

オブジェクトドライバは，タスクフォースとして実行されるサーバである．２章で示したように，タスクフォースは内部に複数のタスクを生成することができる．そのために，オブジェクトドライバの処理も内部で並列処理が可能な形態となっている．

また，タスクとタスクフォースもオブジェクトとして実現されている．２章で示したSVCは，タスクやタスクフォースのメソッドである．

表5　オブジェクトドライバ関係ＳＶＣ

(1)割込み処理関係

___svc_lock_inter	割込みを確保する
___svc_unlock_inter	割込みを解放する
___svc_wait_inter	割込みを待つ

(2)メモリ管理関係

___svc_seg_allocate	メモリセグメントを割り付ける
___svc_seg_duplicate	メモリセグメントを複製する
___svc_seg_divide	メモリセグメントを分割する
___svc_seg_free	メモリセグメントを解放する

(3)オブジェクトドライバ内オブジェクト管理

___svc_enter_file	オブジェクトをカーネルへ登録する
___svc_remove_open_file	カーネルから削除する

(4)ドライバへのメッセージ処理

___svc_get_dev_message	到着したメッセージを受けとる
___svc_send_dev_message	ドライバにメッセージを送る
___svc_dev_open_reply	オープン要求に応答する
___svc_dev_etc_reply	操作要求に対して要求する

(5)呼出し相手の処理

___svc_dev_req_tf_kill	指定されたタスクフォースを消す
___svc_dev_req_task_kill	指定されたタスクを消す

5．おわりに

本報告では，並列処理研究を含めた目的別OSを構築するプリミティブであるSVCセットを提示するとともに，タスクとタスクフォースのプログラミングモデルを提示した．現在，OMICRON V3はユーザレベルでの実用化に向けて，OSのチューンアップ，コマンドインタプリタを初めとする開発環境の移行を行っている．また，これと同時にウィンドウシステムと仮名漢字変換を作成している．これらはオブジェクトドライバとして実現する．

今後の課題としては，次のものがあげられる．

(1) オブジェクトドライバの構成法に関する研究

応用別OSの生成を容易にするには，オブジェクトドライバの構成法についてのガイドラインが必要である．実際に，OMICRON V3のファイルシステムは，モジュール間結合を仮想共有バスとして抽象化したソフトウェアバス [3] で実現されている．

(2) 高水準ユーザライブラリの開発

本報告のSVCセットは，プリミティブセットといってよい．実際に，ユーザがプログラムを作成する時は，より抽象化して使いやすいライブラリを開発する必要がある．

(3) OS作成キットの構築

目的別のOSを容易に作成するために，OS作成キットを構築する．これは，カーネルとOS部品群，OS作成支援系からなるものである．

(4) SVCセットに関する評価

複数の応用研究に対してこのプリミティブセットを適用し，我々が示したSVCセットが，プリミティブセットとして妥当性を持っているか，という点について評価する．

参考文献

[1] 高橋：“研究プロジェクト総説：OS／omicronの開発”，情報処理学会オペレーティングシステム研究会，39－5，1988．6．

[2] 岡野他：“OS／omicron第3版の設計と実現”，情報処理学会オペレーティングシステム研究会，45－11，1989，11

[3] 横関他：“OS／omicron第3版ファイルシステム内部アーキテクチャの設計と実現”，情報処理学会　オペレーティングシステム研究会，47－2，1990．6．

[4] 並木他：“並列／分散処理研究指向のOS“OS／omicron””，情報処理学会計算機アーキテクチャ研究会，83－26，1990．7．