システムソフトウェアと
オペレーティング・システム 70－7
（１９９５．８．２２）

オブジェクト指向のオペレーティングシステムの構成について

白川洋充† 佐藤 睦† 淡誠一郎† 明石 創††

†近畿大学理工学部経営工学科

††日本サン・マイクロシステムズ システム株式会社 応用技術本部

ネットワーク環境が整備され，分散処理が広く一般的になってきた現在，オペレーティングシステムが分散に対応しなければならないという要求はますます強くなっている．そのため多くの分散オペレーティングシステムが開発されてきた．一方，マルチメディアへの応用などに使用するためオブジェクト指向のオペレーティングシステムが最近注目されている．

分散とオブジェクトという二つの概念は別の物であるが，オブジェクトと分散の概念は切り離しては考えられなくなってきた．このような基本的な立場をとった場合オブジェクト指向のオペレーティングシステムはどのような機能をもたなければならないかということについて考察する．

Building an Object-Oriented Operating System

Hiromitsu Shirakawa† Mutsumi Satoh† Seiichiro Dan† Hajime Akashi††

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

††System & Application Engneering Division,
Nihon Sun Microsystems
SBS Tower 4-10-1, Yoga, Setagaya-ku, Tokyo, 158 Japan

Advances in multimedia technologies necessitate the development of object-oriented operating systems. The technologies of multimedia are heavily based on distributed computing system. Recently two methodologies are combined together to make a network objects system. Following this observation we propose the design issues of object-oriented operating systems.

1 はじめに

ネットワーク環境が整備され，分散処理が広く一般的になった今日多くの分散オペレーティングシステムが開発されている [1] [2] [3] [4] [5]．一方，オブジェクト指向のオペレーティングシステムが最近注目されている．分散オペレーティングシステムで採用されたマイクロカーネルの技術の進歩とともにオブジェクト指向のオペレーティングシステムの研究は急速に進んだ．従来のオペレーティングシステムの多くの機能はマイクロカーネルの外にサーバとして置かれ，オペレーティングシステムのモジュール化が行われた．このモジュール群はオブジェクトで実現された．そもそも，オペレーティングシステムをオブジェクト指向にするのはオペレーティングシステムをモジュール化するというのが目的であり，９０年はじめに盛んに行われた．しかし，オペレーティングシステムをオブジェクト指向プログラミング言語で記述し，オペレーティングシステムをオブジェクト指向の概念で構造化するというだけにとどまっている [6]．オブジェクト指向のオペレーティングシステムが注目を浴びている最大の要因にマルチメディアの急激な流行があげられる．オペレーティングシステムがマルチメディアに対応しなければならないためにリアルタイム処理に対する要求が課せられるのみならず，従来のテキストだけでなく音声，画像，動画を扱わなくてはならなくなったため，これらのメディアを統一的にオブジェクトとして扱えるようにするという目的である．

分散とオブジェクトという二つの概念は別の物であるが，これらの概念は切り離しては考えられない．オブジェクトは直接取り扱えず，method を通してしか扱えない．オブジェクトがノード内にあるか，あるいはノード外にあるかをユーザが意識しなくてもオブジェクトの method を直接呼び出すか，あるいは遠隔呼び出しことによってオブジェクトを扱うということが必要になってきた．すなわちオブジェクトはネットワークオブジェクトであるというとらえ方ができる．DEC の SRC の A. Birrel らは RPC を拡張しネットワークオブジェクトを取り扱えるようにした [7]．この考え方は重要で，今後さらに分散とオブジェクトを同じ観点で議論できるように発展させる必要があると考えられる．

さて，以上のことがらをコンセプトの基本とするとオブジェクト指向のオペレーティングシステムはどのような機能をもたなければならないであろうか．一般に分散システムがサポートしなければならない透過性は次のようなものである．

- アクセス透過性 (Access transparency)　局所か遠隔かの区別をしない．
- 位置透過性 (Location transparency)　オブジェクトがどこにあるかを意識しない．
- 移送透過性 (Migration transparency)　性能の改善あるいは耐故障のためにオブジェクトを移送する．
- 同時実行透過性 (Concurrency transparency)　共有オブジェクトを干渉なく並行にアクセスできる．
- 複製透過性 (Replicated transparency)　オブジェクトの有用性 (availability) を高める．
- 故障透過性 (Failure transparency) システムの冗長性を高める．

これらの機能をオブジェクト指向システムにあてはめると次のようになる．アクセス透過性はオブジェクトが「どこにあるのか」,「どのような状態で存在するか」,「どこで実行するのか」ということを性能面から判断しなければならないことを示している．この問題はオブジェクトの格納方法，オブジェクトの移送の問題，あるいは遠隔手続き呼び出しの実現と相俟って非常に困難な問題である．位置透過性はオブジェクトを UID を使って操作する必要がないことを意味する．UID はオブジェクトを格納するときにのみ必要である．そのため，アプリケーションレベルの名前でオブジェクトが操作できればよい．移送透過性はカーネルを含めてすべてのオブジェクトが他のノードに移送できることを意味する．これは最近問題になっている可搬性 (mobile) とも大いに関係する．この概念は計算機が移動した場合にも他のオブジェクトとの連携を保てるかという問題である．オペレーティングシステムにおいてはさらに重要な問題がある．計算機が移動しなくてもユーザが移動した場合，もとのサイトの環境を完全に移す問題 (teleporting) と計算機の故障などでカーネルごと他の計算機に移送し計算を続行する場合がある．同時実行透過性はオブジェクトを共有する重要な概念である．複製透過性は次ぎに説明する故障透過性とともにオペレーティングシステムを耐障害性の上からも必要な事柄である．最後に故障透過性は故障部分をいかにマスクするかが問題で，上に述べた移送透過性と複製透過性の機能があってのみ可能である．

このようなオペレーティングシステムが実現できると本当のネットワークオブジェクトが実現できる．

本稿はこれらの機能のうちで故障透過性をとりあげ，耐障害性機能をオペレーティングシステムを実現するにはどのようなオブジェクトを導入すれば可能であるかを議論するものである．

2 システム構成

耐障害性機構を備えたオペレーティングシステムを実現するのが本論文の目的である．ここでは今後の議論を展開するための前提となるシステムの構成について定義する．

2.1 オブジェクトの定義

システムには様々なオブジェクトが存在する．本システムでは直交した永続オブジェクト (orthogonal persistent object) を扱う．ここで，直交した永続オブジェクトとはあらゆる種類のオブジェクトを一様に操作することを意味する．すなわち，オペレーティングシステムを構成するキューや構造体のような低レベルのデータタイプも永続性を持たせ，かつプロセスや画像データのようなオブジェクトも永続性をもたせ，どのようなオブジェクトも一様に操作できるようにすることを意味する．

これらのオブジェクトは大きく次の三つに分けることができる．

(1) システムオブジェクト

(2) ツールオブジェクト

(3) アプリケーションオブジェクト

システムオブジェクトは，従来のマイクロカーネルの外に存在する種々のサーバに対応している．ツールオブジェクトは，オペレーティングシステムを構成する一般的なデータ構造を抽象化したオブジェクトである．アプリケーションオブジェクトは，ユーザが作成するオブジェクトである．ユーザがアプリケーションを作成する場合は，ツールオブジェクトのクラスを使いシステムオブジェクトに準じて作成する．オペレーティングシステムのツールを使うためオペレーティングシステムの追加，変更もアプリケーションオブジェクトを作成することになる．

2.2 マイクロカーネル

カーネルの持っている機能を独立させ，それらをカーネルの外部に配置することによりカーネルを極小化する．カーネルの外部に配置された機能は独立したシステムオブジェクトとして存在する．

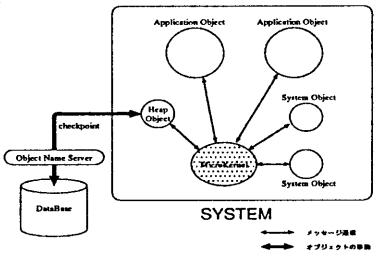

図 1 システム構成

2.3 データベース

データベースには，オペレーティングシステムを構成する全オブジェクトが登録されている．オブジェクトには，グローバルで一意的な識別子 (UID) としてディスクブロック (24bit) とマウント機器 (8bit) とノード情報 (32bit) の計 64bit のデータを付加する．このようにすることで，識別子によって直接ディスクのオブジェクトにアクセス可能となる．

このデータベースはオペレーティングシステムからは，サイズ p のスワップ領域の集合体として見える．ここで $p \leq m$ とする．m は最大のスワップ領域サイズとする．本システムで設計するオペレーティングシステムは，データベースがスワップ領域の代わりをするので，従来のスワップ領域は持たない．また，このデータベースは安定記憶 (stable file) と仮定する．

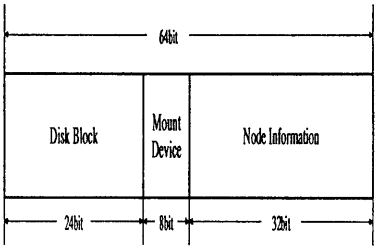

図 2 オブジェクトの識別子 (UID)

2.4 オブジェクトネームサーバ

データベースに登録されているオブジェクトは，オブジェクトネームサーバによって管理される．

2.5 アドレス変換部 (Pointer Swizzling)

アドレス変換部は，オブジェクト間の参照に係わるアクセス時間を軽減するために UID 形式の形で表されている永続ポインタ (persistent pointer) をメモリのポインタあるいは単にアドレスに変換することをいう [8] [9] [10].

データベースに格納されている永続オブジェクトを従来のメモリマネージングユニット (MMU) 経由でメモリ上にマッピングできる形式に変換する．変換方法についての詳細は，次章で述べる．

3 オブジェクトの変換

前節で述べたようにオブジェクトはメモリ上に存在する場合とデータベースに格納されている場合ではデータの形式が異なる．後者の永続オブジェクトのポインタはオフセットあるいは UID でなければならない．また，データベースの永続オブジェクトは異なったマシンに転送されることを考え，マシンに依存しない形式であることが必要である．

3.1 アドレス変換

オブジェクトがデータベースからメモリに読み込まれる時には，以下の方法でアドレス変換が行われる．

まず，オブジェクトのヘッダ部に，どのオブジェクトを参照しているかという情報と，そのオブジェクトのページサイズ n($n \leq m$) を記述する．マルチメディアデータの場合には，連続したページをメモリに配置 (allocate) するために何ページ使用するかを記述しておくことが必要である．また，図 3 にデータベースに存在するオブジェクトを示す．各オブジェクトに対して次に示すようなテーブルがメモリの上に作られ以下の情報を記述する．

- オブジェクトのアプリケーションレベルの名前とメモリロケーション情報
- オブジェクト間の参照関係
- オブジェクトごとのページテーブルとページ情報

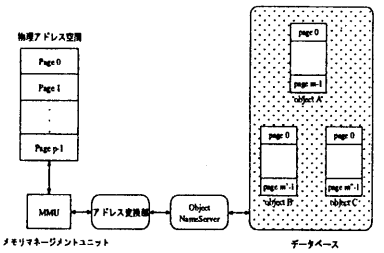

図 3 オブジェクトの変換

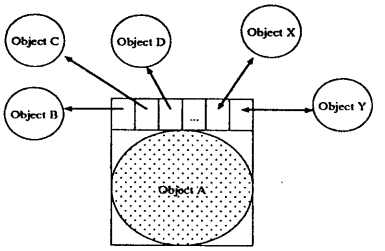

図 4 オブジェクト間の参照

アドレス変換は，オブジェクトがメモリにある場合にはアプリケーションレベルの名前と UID の関係からアプリケーションレベルの名前と実メモリのロケーションの関係に書き換える．このアプリケーションレベルの名前と UID の関係はオブジェクトネームサーバによって管理する．オブジェクトの 0 ページは，オブジェクトの情報を格納するために使用する．

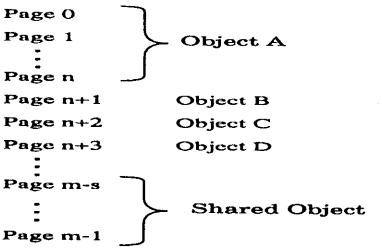

図 5 オブジェクト参照時のページテーブル

ここで，ページ数 n のオブジェクト A がオブジェクト B, C, D を参照しているとする．この時のオブジェクト A のページテーブルは，図 5 のようになる．なお，ページテーブルのサイズは m である．

オブジェクト A のページ 0 には，ページ n+1,

n+2, n+3 がそれぞれオブジェクト B, C, D に対応していることを書いておき，そのページにアクセスがあるとページフォールトが発生するようにしておく．ページフォールトが発生するとコンテキストが切り替わる．参照されたオブジェクトが既にメモリ上にある場合には，そのオブジェクトが配置されているページ番号を，そうでなければメモリ上にないことをオブジェクト A のページテーブルに記述しておく．このようにすることで，既にメモリ上にあるオブジェクトは共有できるようになる．ページテーブルの最後に配置されている共有オブジェクトとはメモリ常駐のオブジェクトである．これは各オブジェクトに共通して置く．ただし，この共有オブジェクトからは他のオブジェクトを参照しないものとする．このオブジェクトを参照するときはコンテキストが切り替わらない．

3.2 ヒープオブジェクト

オペレーティングシステムの状態，オブジェクトの管理情報，オブジェクト間のメッセージログ，さらにすべてのオブジェクトの状態はヒープ領域にオブジェクトとして管理，保管する．これが次に述べる耐障害性機構をオペレーティングシステムに備えるために使用される．

4 耐障害性機構

システムの信頼性を向上させるための手法の一つに，チェックポイントとロールバックがある．

4.1 諸定義

チェックポイント・ロールバックアルゴリズムは以下の条件をもとに論じられることが多い．

- チェックポイント取得の単位はプロセスであり，これらのプロセスは揮発記憶と安定記憶を持つ．
- プロセス間に共有メモリは存在せず，プロセス間の通信はメッセージパッシングで行う．
- プロセス間通信で使用される通信チャネルは無故障かつ無限長の FIFO バッファである．
- プロセスの故障は他のプロセスに影響を及ぼすことはなく(停止故障)，検出可能である．

本システムにチェックポイント・ロールバックアルゴリズムを導入するにあたって以下の条件を設ける．

- チェックポイントの取得単位はオブジェクトである．
- 共有メモリは存在せず，通信はメッセージのみで行う．
- 故障としては停止故障のみを考え，ロールバック時には故障しないものとする．
- 状態を保存しておくデータベースは故障しないものとする．

通信機構でメッセージのチェックを行うことにより，メッセージの順序の保証を行うことは可能である．

4.2 一貫性

オブジェクトの集合体としてシステムが動作している場合に，あるオブジェクトに故障が発生した場合には様々な処理が必要となる．耐障害性を導入した場合には，故障したオブジェクトが修復された時にチェックポイントをロールバックしてその時点から再スタートできるようにシステムの一貫性を保つ必要がある．

一貫性の保持されたシステムとは以下の条件を満たすものである．

(1) 送信側オブジェクトで未送信，受信側オブジェクトで受信済みのメッセージが存在しない．

(2) 送信側オブジェクトで送信済み，受信側オブジェクトで未受信のメッセージが存在する場合にはそのログが残されている．

(3) チェックポイント更新完了後に次のチェックポイント更新が行われる．

図 6に一貫性を保持したシステム状態の例を示す．この図では，メッセージ m3 は，オブジェクト A のチェックポイント cp1 の取得前に送信され，オブジェクト B のチェックポイント cp2 の取得後に到着している．また，m2 も cp3 取得前に送信され，受信は cp2 取得後となっている．この状態で，オブジェクト B が cp2 から再スタートした場合，m3 はオブジェクト A では cp1 の前の処理なので送信済みとなっているが，オブジェクト B では cp2 の時点ではまだ m3 は受信済みにはなっていない．m2 についても同様である．このような場合に備えて，上記 (2) のとおりログを保存しておき，オブジェクト B が cp2 から再スタートした場合には，保存してあったログを使用することにより，システムの一貫性は保たれる．

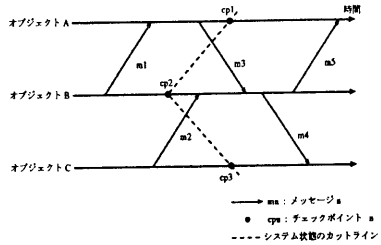

図6　一貫性を保持したシステム状態

次に，図7に一貫性が保持されていないシステム状態の例を示す．この図では，メッセージm3は，オブジェクトAのチェックポイントcp1の取得後に送信され，オブジェクトBのチェックポイントcp2取得前に受信されている．このような状態でチェックポイントを取得していると，オブジェクトAをcp1から，オブジェクトBをcp2から再スタートした場合，m3はcp2の時点で受信済みとなっているにも関わらず，cp1では送信済みとはなっていないので，m3はオブジェクトAから再送されるが，オブジェクトBでは受信済みとなっているために，処理を行うことはできない．このように，送信側オブジェクトで未送信，受信側オブジェクトで受信済みのメッセージが存在するとシステムの一貫性を保持することはできない．

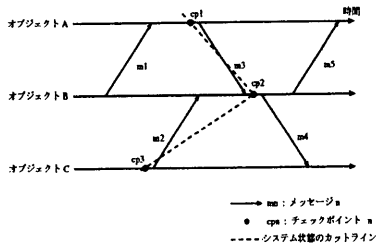

図7　一貫性を保持していないシステム状態

したがって，図6のような状態でチェックポイントが保持されていれば，システムが故障しても回復させることができる．しかし，図7の状態ではシステムが故障しても情報が矛盾してしまっているのでシステムを回復させることはできない．

4.3　データベースを用いたチェックポイント・ロールバック

システムの状態を保存するチェックポイント取得の単位は，様々なレベルで考えられるが，本システムではオブジェクトの集合によってオペレーティングシステムが構成されているので，オブジェクトを単位とする．

4.3.1　取得方法

チェックポイントの取得方法は大きく自律型と協調型に分類される．自律型とは個々のノードが非同期でチェックポイントを取得する．個々のノードが独立してチェックポイントを取得するので，ノード単位でチェックポイントの間隔を決定することができる．一方協調型は，全ノードがシステムの一貫性を保持するために同期してチェックポイントの取得を行う方法で，この協調型ではシステムの一貫性を保持しやすいが，全ノードで同じタイミングでチェックポイントを取得するため，柔軟なチェックポイントは行えない．このシステムでは，オブジェクト単位でチェックポイントを行い，状態を保存するというシステムを導入するため自律型を採用した．

4.3.2　更新

チェックポイントは，ヒープオブジェクトにオブジェクト記述子（従来のプロセス記述子に相当），スタック，メッセージログを間欠的にストアし，システムがこのヒープにストアされたオブジェクトの状態に一貫性があると判断した段階でをデータベースにチェックポイント要求のメッセージを送信し更新する．

4.4　故障

ノードが故障するとチェックポイントの更新に影響が出るので速やかに故障は検出されなければならない．故障の検出はメッセージ送受信のタイムアウトで行うので，チェックポイントの更新時には必ず故障は検出される．一度故障とみなされたノードは自ら回復を宣言するまでは，故障として扱われる．

オブジェクトに故障が発生すると，前回のチェックポイント以降，どの時点で故障したのかを特定するのは不可能である．従って，その間の処理については無効とし，オブジェクトが修復されたときに再実行する．

故障したオブジェクトについては前回のチェックポイントをロールバックして再実行すれば問題ない

が，稼働中のその他のオブジェクトでは無効とされた処理の影響を受けている可能性がある．これは他のオブジェクトからの干渉を受けた（メッセージを受信した）場合である．それぞれが独立したオブジェクトとしてチェックポイントを取っている場合は，故障によって失われた実行から影響を受ける実行部分のみをロールバックすればよい．従って図8のような場合には，故障したオブジェクト B のチェックポイントおよび受信メッセージ m2 の内容がヒープオブジェクトに保存されているとする．オブジェクト B はこの保存情報を用いて m4 を送信する所まで再実行可能である．オブジェクトはそれぞれが独立しているので，再実行時の m3，m4 の内容は故障前の実行時と同一である．したがって，m5 の内容がオブジェクト B のヒープオブジェクトに保存されていたとすると，オブジェクト A はロールバックすることなく，オブジェクト B からの要求に応じて m5 を再送信することができる．

しかし m5 と m6 の受信順序の情報は失われてる．したがって，再実行時の m4 の送信より後のオブジェクト B のメッセージ送信は故障前と異なる可能性があるが，これはメッセージの受信順序を記録しておくことで解決できる．

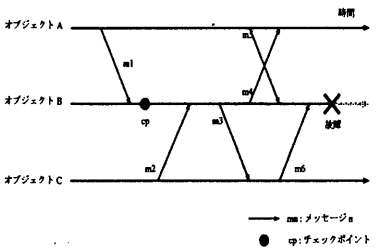

図 8　ロールバックが必要な故障

4.5　耐障害性機構のシステム

本システムでは，オブジェクトは全てデータベースに登録されている．そのオブジェクトはシステムのメモリ上に読み込まれても不変であるため，チェックポイントをする必要はない．また，可変データは全てヒープオブジェクトに格納される．よってチェックポイントによって保存されるべきデータはヒープオブジェクトによって管理されているオブジェクトのみである．このオブジェクトに識別子を付加してデータベースに格納するだけでよい．チェックポイントは，コンテキストスイッチが切り替わると同時に発行される．

4.6　応用

耐障害性機構の応用としてテレポートを考える．テレポートとは，図9のようなものである．例えば，あるサイトでマシンを使用していたが，ネットワークで接続されている他のサイトのマシンで作業する必要が出た場合に，いままでの作業環境を転送し，マシンの位置を意識せずに作業を継続することができることである．

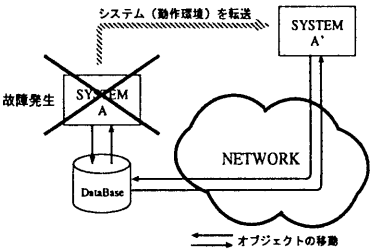

図 9　テレポート

これは，本システムがデータベースに登録されているオブジェクトをネットワークを経由してシステムを構築するという構成をとっているのと，チェックポイントによって可変データを全てデータベースに登録してあるからである．このことから，環境の移動は，移動前のマシンのシステムを構成していたオブジェクト群とそれらのチェックポイントによってデータベースに格納されたデータをそのままネットワークを介して移動先のホストに送ればよい．このような応用は今後の移動体の計算機システムのオペレーティングシステムにとって重要であると考えられる．

5　おわりに

本論文では，分散とオブジェクトという二つの概念は個別には考えられないことを説明し，この立場のもとでオブジェクト指向のオペレーティングシステムはどのような機能を持たなければならないかを考察した．これらの機能の中でも特に重要な耐障害性機構を備えたオブジェクト指向オペレーティングシステムに関する検討を行った．また，今後移動体通信に使われる重要なテレポーティングについても考えた．なお，この論文は [11],[12] をもとにしていることを触れておく．

参考文献

[1] M. J. Accetta, R. V. Baron, W. Bolosky, D. B. Golub, R. F. Rashid, A. Tevanian, Jr., M. W. Young: Mach, A New Kernel Foundation for UNIX Development, Proc. of Summer USENIX Conf., pp. 93-113, Jun. 1986.

[2] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren: Amoeba, A Distributed Operating System for the1990s," IEEE Comp. Mag., vol. 23, no. 5, pp. 44-54, May 1990.

[3] Guillemont, J. Lipkis, D. Orr, and M. Rozier: Chorus systems, A Second-Generation Micro-Kernel Based UNIX; Lessons in Performance and Compatibility, Proc. of Winter USENIX Conf., pp. 13-22, Jan. 1991.

[4] M. Gien: From Operating Systems to Cooperative Operating Environments, Proc. 10th Anniversary Int'l. UNIX Symp., pp. 1-10, July 1992.

[5] D. R. Cheriton, G. R. Whitehead, and E. W. Sznyter: Binary Emulation of UNIX using the V Kernel, Proc. of Summer USENIX Conf., pp. 87-96, June 1990.

[6] Hiromitsu Shirakawa and Eiji Okubo: When Object-Oriented Operating System is Time Critical, Proc. EUROMICRO'92 Workshop on Real-Time Systems, pp.54-59, 1992.

[7] Andrew Birrell, Greg Nelson, Susan Owicki, and Edward Wobber: Network Objects, DEC SRC Res. Rep. 115, Feb. 1994.

[8] J. Eliot B. Moss: Working with Persistent Objects: To Swizzle or Not to Swizzle, IEEE Trans. Software Eng., 18(8), pp.657-673, 1992.

[9] Vivek Singhal, Sheetal V. Kakkad, and Paul R. Wilson: Texas: An Efficient, Portable Persistent Store, Proc. Fifth Int'l. Workshop on Persistent Object Systems, pp.11-33, 1992.

[10] Paul R. Wilson and Sheetal V. Kakkad: Pointer Swizzling at Page Fault Time: Efficiently and Compatibly Supporting Huge Address Spaces on Standard Hardware, Proc. Second Int'l Workshop on Object Orientation in Operating Systems, pp. 364-377, 1992.

[11] 加藤暢敬: オペレーティングシステム Theta における耐障害性機構の設計, 立命館大学大学院理工学研究科修士論文, 1993.

[12] 明石　創: 耐障害性機構を備えたオブジェクト指向オペレーティングシステムの設計, 立命館大学大学院理工学研究科修士論文, 1994.