■7 群（コンピュータ -ソフトウェア）－3 編（オペレーティングシステム）

9 章 OS の構成法と仮想計算機

（執筆者：吉澤康文）［2013 年 2 月 受領］

■概要■

ハードウェアとの接点をもち，またアプリケーションソフトウェアとの接点を提供する OS をどのような構造に設計されてきたかについて本章では解説する．今まで数種類の実装モデルが存在するが，それぞれ利点・欠点を保有して今日に至っている．OS と称していても現代の OS とは概念がかけ離れており，構成という概念のないものは除外している．今後，発展が期待されるが，その開発や理解のためにも資するものと考える．

【本章の構成】

各種の OS 構成法が考案されてきた．ここでは，単一構成方式，マイクロカーネル方式，などについて紙面の関係からその概要を説明する（9-1 節）．次に，仮想計算機システムの基本的な考え方として，仮想計算機が生まれた背景，用途などを解説する．同時に，その動作原理について概観し，具体的なイメージを説明する（9-2 節）．

■7群 － 3編 － 9章

9-1 OSの構成方式

（執筆者：吉澤康文）［2013年2月 受領］

9-1-1 単一構成

OS の設計ではユーザに提供する機能を満たすだけでなく，性能を考慮した設計を行わなくてはならない．また，ソフトウェアの多くは機能拡張やバグの修正などのために人手が加えられることが常であり OS も例外ではない．このため，設計の重要な項目として柔軟な構造にすることがあげられる．例えば，機能を拡張しようと思ったときに容易に対応できるかなどは重要な要素である．このために各種のソフトウェア工学的な手法が用いられているが，ここでは，そのような観点からではなく，OS の機能をどのように構築するかを見ていく．

OS の構成法で問題としているのは，制御の区分けである．2 章では OS はコンピュータ資源のすべてを把握し，プロセスに分配する役割があることを述べた．このために，CPU の特権状態（2 章 2-1-5 項）で実行しなければならない部分がある．代表的な例は入出力の実行である．そこで，最も単純な OS の構成法は**図 9･1** に示すように OS の機能はすべてが特権状態で実行させる方法である．この方式をモノリシック（Monolithic）方式という．つまり，OS 機能全部が一枚岩のような単純な構造になっているのである．初期の OS はほとんどすべてがこのような方式で実現されていた．

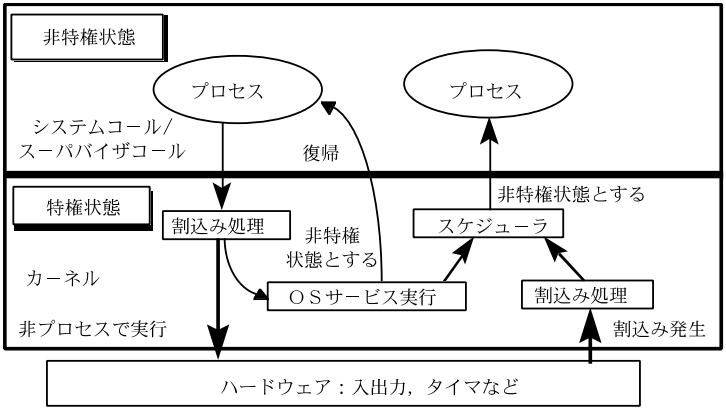


図 9・1　モノリシックなカーネル構成

モノリシック構成のカーネルでは，図 9･1 に示すように，ユーザプロセスがシステムコールを実行すると割込みとしてカーネルに制御が渡り，カーネルはすべての処理を特権状態で実行する．そこでは，OS のサービス機能を実行した後に，プロセススイッチを不要とするか否かによって直接ユーザプロセスに制御を戻すかスケジューラに制御を渡すことになる．いずれにしてもユーザプロセスに制御を渡すときは非特権状態にして制御を渡すことになる．このようなシステムコールによるプロセスの要求により，ハードウェアに対する操作を行う必要が生じるので，特権命令を実行する必要があるので特権状態での実行部分が必要となる．

特権命令を実行すると，入出力やタイマなどのように割込みが発生するが，モノリシック方式では，その場合のすべての処理を特権状態で実行する．

モノリシック構成のカーネルの利点は OS の全機能を特権状態で実行させるので，単純であることであるが，特権状態でのプログラムの実行は，すべてのメモリ領域に対してアクセスが可能であるため，OS にバグがあったときにシステムダウンにつながる可能性が高くなる．また，特権状態で実行させるために，割込みを禁止してすべての処理を実行することになったり，割込み禁止の範囲が長くなったりする可能性が高い．したがって，他の割込みが発生していても待たせることになり，割込みに対する応答性能が悪いということになりかねない．

モノリシック方式の例として**図 9･2** に UNIX 系 OS を示す．多くの UNIX 系 OS はこのような方式であり，UNIX カーネルの中にすべての OS サービスプログラムが組み込まれており特権状態で実行される．

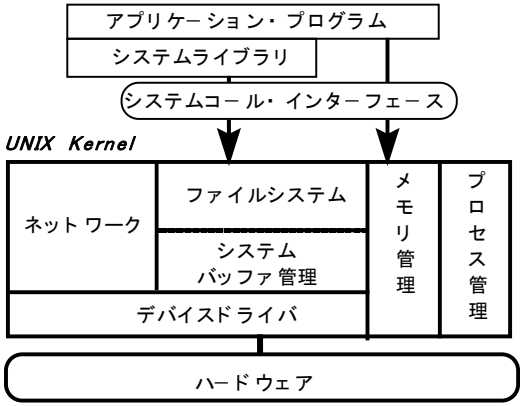


図 9・2　UNIX 系カーネル構成の例

9-1-2 マイクロカーネル

OS の構成については多くの研究が行われ，各種の提案がなされている．近代の OS は用途が極端に広くなっていることもあり大規模化し，開発する場合にはいくつもの問題点を含んでいる．代表的な開発での問題は以下のとおりである．

(1) 膨大な開発工数を削減したい．
(2) ハードウェアの進歩に追従しなければならないので，機能追加を簡単にしたい．
(3) 新しいハードウェアのサポートを行う必要がある．
(4) ハードウェア依存度の高い部分と論理的な処理を分離したい．
(5) OS のバグがシステムダウンとならないフォールトトレラントを実現したい．
(6) 割込み禁止区間は最小限とし，割込みの応答性能を高めたい．
(7) 記憶保護の観点から非特権状態で実行できる部分を増やしたい．
(8) 複数の OS 機能を同時に 1 台のコンピュータ上に実現したい．

このような状況から，なるべくハードウェアに依存した部分を小さなカーネルとして作成

し，論理的な OS 機能はプロセスとして実現するような方式が提案されている．それが，マイクロカーネル（Microkernel）である．**図 9･3** にはマイクロカーネルの考え方で構成する OS の一例を示した．このようなアプローチをとっている代表的なカーネルは，米国カーネギーメロン大学の Mach, フランス INRIA の Chorus, 米国 OSF（Open Software Foundation）の OSF/1 などであり，これらは商用化された OS として利用されている．

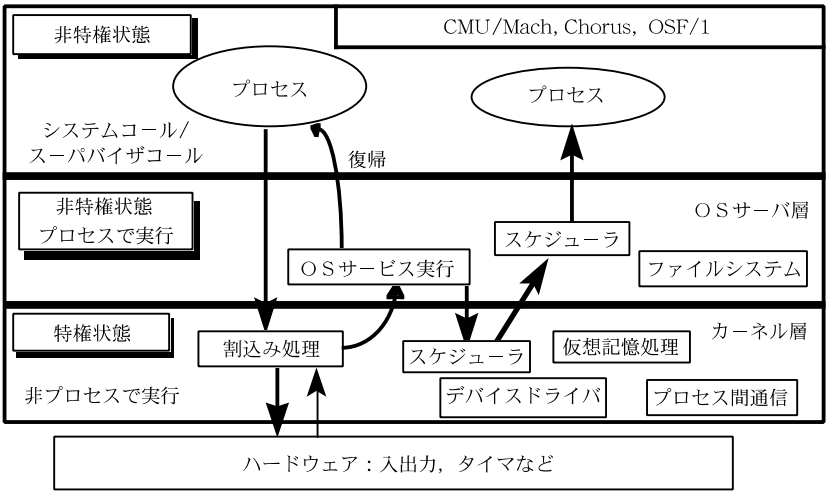


図 9・3　マイクロカーネルによるカーネル構成法

この構成では，3 層の構造をもっている．一番下はカーネル層でありハードウェアとの接点であり，最も基本的な機能しか保有していない．この部分だけが CPU の特権状態で実行されハードウェアを抽象化して上位の層に情報を提供している．ここでは，割込み処理，プロセススケジューラ，仮想記憶管理，一部のデバイスドライバなどの機能がある．中間の層は OS サーバ層とでもいうべき役割をもった機能を配置する．この層には，モノリシックなカーネルのもつ機能を配置し最上位の層にあるアプリケーションプログラムの実行を行うプロセスに対するサービスを行う．例えば，UNIX のファイルシステムやプロセス間通信機能，ネットワークサービスなどである．

第 2 層の存在は，機能マシンとして各 OS が提供しているシステムコールやファイルシステムのインタフェースを実現している．したがって，マイクロカーネルでは，この層を OS サーバ層として各種の OS 機能を用意することができる．例えば，UNIX のサービス機能，DOS のサービス機能，Macintosh OS のサービス機能などである．このように 1 台のコンピュータに複数の OS を実現することが可能となるので，このことをマルチパーソナリティ（Multi-personality）と呼ぶことがある．つまり，OS を一つの人格を所有した物と抽象しているのである．その例を**図 9･4** に示した．

マイクロカーネルでは OS サーバ機能もプロセスとして実行しており，アプリケーションプログラムの実行と区別はない．プロセス間の通信で機能の分担を行うのであるが，その際に，マイクロカーネルが仲介の役割を行うことになる．それがシステムコールとなってマイ

クロカーネルに制御が渡ることになる（図 9･3 を参照）.

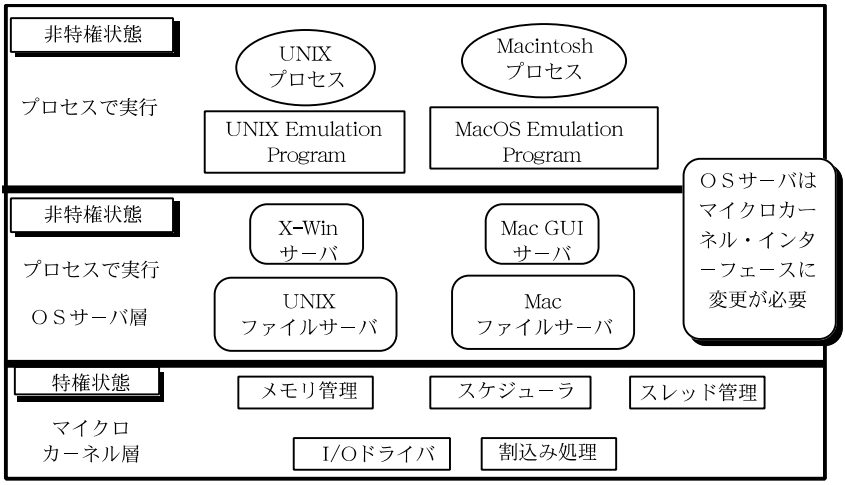


図 9・4　マイクロカーネルによる複数 OS の実現法

マイクロカーネルによる OS サーバ方式を採用すると，アプリケーションのプロセスがサーバに対する要求はプロセス間通信でよいし，またマイクロカーネルと OS サーバの通信も同様にプロセス間通信となる．したがって，プロセス間通信をネットワークも含ませて実現しているならば，各プロセスが 1 台のコンピュータ上に存在する必要がなくなる．このことから，OS 機能を分散させた分散 OS が可能になる利点がある．

最後に，このような構成を作ることにより OS 開発の課題であるいくつかの問題が解決されてきたのであるが，新たな問題としては機能の呼び出しをすべてプロセス間通信で行うために，そこにマイクロカーネルの介在が生じることでオーバヘッドが増大することである．信頼性や OS のモジュール化開発，そしてユーザにはマルチパーソナリティなどの利点も提供できるようになったが，このような問題もある．

9-1-3 その他構成法

モノリシックな OS では特権状態ですべての実行を行う．このため OS の保守，デバッグ，などの障害になり，また OS のバグがシステムクラッシュになる可能性が高いなどの欠点をもっている．したがって，非特権状態で実行できる OS 機能を極力増やした方がよい．

このよう考えると，ファイルシステムはプログラマに提供しているファイルの論理的なインタフェースの部分と物理的な記録媒体の操作を行うデバイスドライバ部分から構成されているので，論理的な部分は非特権状態で実行してもよいことになる．そこで，**図 9･5** に示すような方式が考えられる．これは，IBM 社の MVS などが採用している方式である．図のなかの数字はユーザプログラムからファイルに対する読込みなどの要求が出たときの処理の順番である．

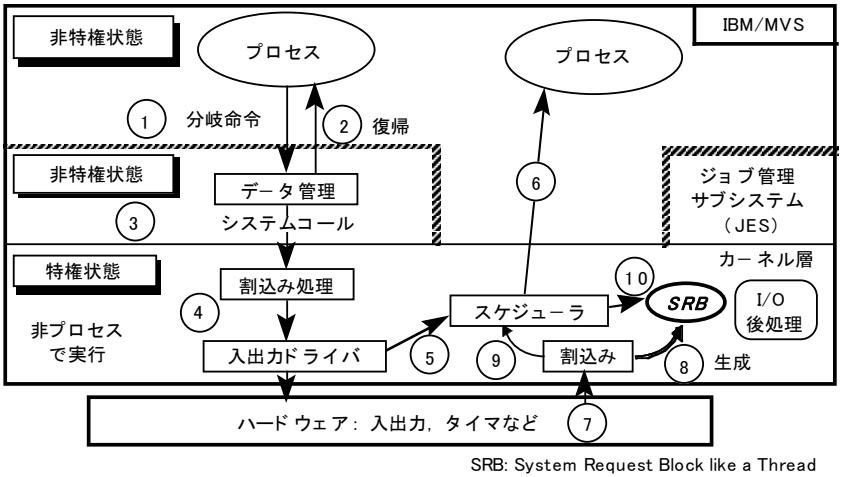


図 9・5　IBM/MVS のカーネル構成法

ユーザプログラムからは，読込み要求が出るが，これは分岐命令でデータ管理のプログラムに制御が移動する（①）．この部分はシステムコールのような割込みにはならない．データ管理は要求を分析した結果，システムコールを実行せずに済む場合は適切な処理を行ってユーザプログラムに制御を渡す（②）．例えば，読込みバッファ内に要求されたレコードが存在するならばデータをユーザプログラムにそのまま渡すことができるときなどである．

システムコールを実行すると（③）割込みになり入出力ドライバに制御が渡る（④）．ここでは，入出力装置を起動するが，完了すると入出力完了までの間当該プロセスは動作しないので，制御をスケジューラに渡す（⑤）．スケジューラは入出力要求のプロセスかその他のプロセスをスケジューリングアルゴリズムに従って選択しディスパッチする（⑥）．

入出力が完了すると，割込み処理に必要な情報を短い処理で取得し（⑦），割込みの処理をプロセスとして実行させるためにスレッドのような軽量プロセスを生成する（⑧）．そして，プロセススケジューラに制御を渡す（⑨）．一般的に，割込み処理のようなスレッドは処理優先度が高いので，スケジュールされて入出力の後処理がなされる（⑩）．このようにして入出力処理が完了すると，入出力後処理では，最初に入出力要求したプロセスに入出力完了事象を通知する．この場合，当該プロセスが入出力待ち状態になっているならば，待ち状態を解除し実行待ち状態にする．

■7群 － 3編 － 9章

9-2 仮想計算機システム

（執筆者：吉澤康文）［2013年2月 受領］

9-2-1 基本的な考え方

ある一つのコンピュータアーキテクチャ上に複数の OS を使わなければならない場合がある．例えば，以下のようなときである．

(1) 古い OS から新 OS に移行する経過措置として
(2) 既存システムを使用しながら新システム開発をしているとき
(3) OS 依存のアプリケーションを複数動かさねばならないとき
(4) 既存システムを使用しながら新しい OS を開発するとき
(5) 異種の OS 機能を試行してみたいが既存のコンピュータ機能を停止できないとき
(6) 複数サーバを抱えたサイトで省エネならびに省力化による運用コストを下げたい場合

本来ならば複数台のコンピュータ（サーバ）を導入することが望ましい．しかし，大型コンピュータなどでは 1 台のコンピュータが高価であり場所や電源の都合などから不可能なことがある．そこで，コンピュータ資源をすべて論理化して仮想的なコンピュータを複数台提供できる機構があることが望まれる．仮想計算機（VM：Virtual Machine）とはこのような考えから生まれた．

9-2-2 動作原理

1 台のコンピュータの上に論理的な複数のコンピュータをソフトウェアにより作り，各々の論理化されたコンピュータに OS をローディングして実行させる方式を仮想計算機と呼ぶ．この場合，OS は同一でも別々であっても構わない．コンピュータ資源の 3 要素である，CPU，メモリ，入出力装置を仮想化（論理化）することで物理的な制約から解放される利点がある．仮想化するソフトウェア（VMCP：Virtual Machine Control Program）は特権状態で実行され，また割込み処理も行うことになるが，その他の仮想計算機でのプログラム実行は非特権状態で実行される．つまり，仮想計算機の OS は非特権状態で実行されることになる．

このような状況から，VMCP は複数の OS の上位に位置する制御プログラムであるため，ハイパーバイザなどと呼ぶことがある．すなわち，従来の OS が担っている実資源管理を VMCP が行い，仮想計算機の OS は仮想的な資源を取り扱うことになるので，従来の OS の考えからすると，VMCP が OS に，VM がプロセスに相当することになる．したがって，VMCP は各 VM をディスパッチする．

仮想計算機の動作原理を説明したのが**図 9･6** である．この図ではある OS の下で実行しているアプリケーションプログラムから入出力要求が発生したときの処理の手順を示している．アプリケーションプログラムからは入出力要求がシステムコールとして出てくる．そこで，割込みが生じるが制御は VMCP に入ってくる．VMCP はシステムコールの発生であるので，このアプリケーションプログラムを管理している OS のシステムコール割込みアドレスを求めて制御を渡す．このとき，論理化された計算機の状態はシステムコール割込み状態であることを記録しておく．このような仮想計算機の状態を記録しておく制御テーブル（VM_BLOCK）を各 VM に用意しておく．これはちょうどプロセス制御テーブルに相当する．

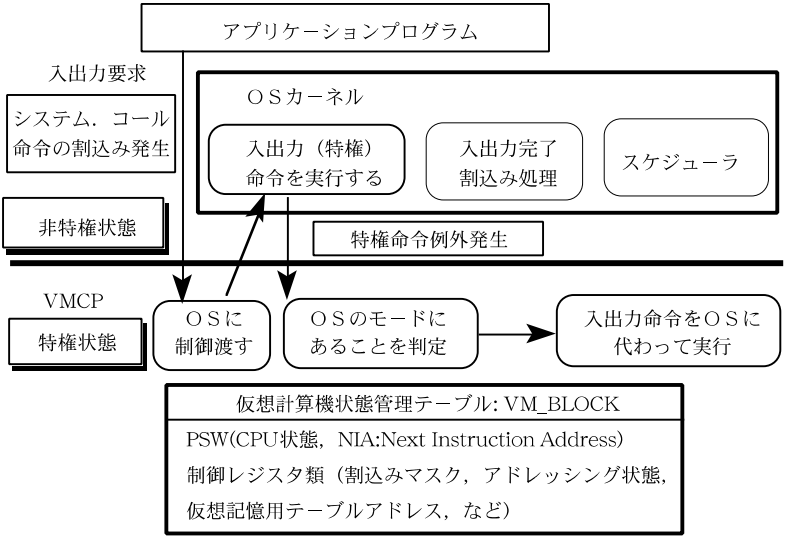
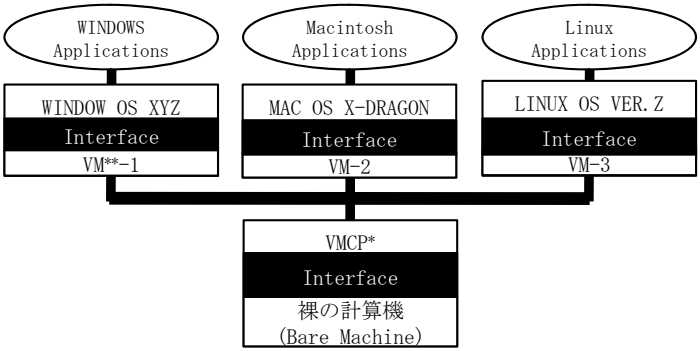


図 9・6　仮想計算機の動作原理

図 9･6 にあるように，各 VM には VM_BLOCK が用意され，その中には，CPU の状態，次に実行するアドレス，各種の CPU レジスタ類が格納されている．したがって，上記の入出力要求は該当する VM の割込み処理ルーチンに仮想的な特権状態で制御が渡ることになる．VM の OS カーネルは制御が渡ると，入出力命令を実行して，アプリケーションプログラムの要求を満たそうとする．しかし，実際には仮想的な特権状態であるので，特権命令例外の割込みが生じることになる．そこで，VMCP に制御が渡ってくるが，VMCP はこの特権命令が仮想的な特権状態から実行されたことを判断し，正当であると判断する．そこで，VM の OS に代わって入出力命令を VMCP が実行することになる．この操作を特権命令シミュレーションというが，仮想計算機を実現する際のオーバヘッドとなり性能の低下要因となる．このため入出力を実行すると，その完了割込みが生じる．VMCP に制御が渡るが，そのとき，割込みがいずれの VM に対する割込みであるかを判別し，あたかも現実のハードウェア上で割込みが生じたかのように VM_BLOCK 内の関連レジスタ，VM に割付けたメモリ上などに情報を書き込み，VM の OS の入出力割込みルーチンに制御を渡す．このようにして，入出力割込みの仮想化を行う．

9-2-3 仮想計算機の例

図 9･7 に仮想計算機の例を示した．同一あるいは同類のコンピュータアーキテクチャを共有する OS を複数同時に 1 台のコンピュータ上で実行することが可能になる．マイクロカーネルのところで述べたマルチパーソナリティはこのような仮想計算機の方式でも実現できる．ここで述べた方式は，OS に一切の変更をせずに済むのが利点であるが，このような VMCP を作りやすいか否かはコンピュータアーキテクチャに依存する．したがって，この方式は利点もあるが難しい点もある．



VMCP*: Virtual Machine Control Program, VM**: Virtual Machine

図 9・7 仮想計算機による複数 OS の実行機構

マイクロカーネルでのマルチパーソナリティの実現は，既存の OS を OS サーバというプロセスで実現するために，マイクロカーネルとのインタフェースに従った設計になる．したがって，既存の OS をサーバとして動作させるには，このインタフェースとなる部分に修正を加える必要がある．

各 OS サーバはプロセスとして実行することになるので，ファイルシステムやメモリ管理，スケジューラなどを複数の OS サーバとして再構成したときは，他のプロセスとの間をマイクロカーネルが提供するプロセス間通信のインタフェースに変更しなければならない．すなわち，従来のようにモノリシックなカーネルならばブランチ命令で制御が渡っていたところをマイクロカーネル経由になるためにオーバヘッドが大きくなる欠点がある．しかし，OS 内の一部の機能にバグがあってもシステムクラッシュに至らない利点やバグの所在が明確になるなどの利点がある．何よりも増して，マルチパーソナリティが実現できるのである．

9-2-4 仮想計算機の問題点と解決策

VM がディスパッチされると実行されるプログラムは OS であったりその OS の下で実行されるアプリケーションプログラムであったりする．ここで問題は，OS が実行しているときである．OS は特権状態で実行しているという仮定で命令を実行するので，特権命令を実行する．例えば入出力操作やタイマの処理である．このような場合，非特権状態から特権命令が実行されると特権命令例外の割込みが発生し，制御が VMCP に渡ってくる．

VMCP は特権命令例外の割込みが生じると，該当命令を実行した VM の VM_BLOCK 内の CPU 状態を確認する．もし，VM が特権状態であるならば，この特権命令は正しく実行されねばならないので，VMCP が特権命令を VM が実行したようにソフトウェアによりシミュレーションする．しかし，VM が非特権状態であるならば，特権命令例外の割込み処理を VM が行えるように VM_BLOCK 内に情報を格納する．**図 9・8** にその処理方法を示した．

ここで問題になるのは VMCP が行わなくてはならない特権命令シミュレーションにある．つまり，本来ならば VM 上のカーネルの実行する特権命令であるので正当に実行されてよいのであるが，VMCP の管理下で実行しているために，割込みが生じること，ならびに特権命

令シミュレーションをソフトウェアで行うことから CPU を消費してしまうことになる．このような問題を解決するために工夫が行われ，図 9･8 に示した論理をマイクロプログラミングによって高速に実現する方式として VMA（Virtual Machine Assist）なるアプローチがとられたりしている．最近は，VMA よりも高速な実行の仕掛けがハードウェアならびにソフトウェアにより実現する方式が考案され実用化されている．

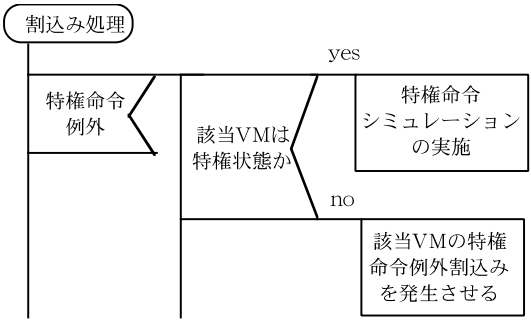


図 9・8　VMCP での割込み処理

9-2-5 VM マイグレーション

ネットワーク，サーバ，ストレージ，アプリケーション，サービスなどの構成可能な計算機資源を共用とし，これらの資源を必要に応じて利用可能とし，かつ最小の管理労力で，場合によってはこれらの提供者同士が協力することで提供するコンピューティングモデル（Computing Model）がクラウドコンピューティング（Cloud Computing）と呼ばれている．つまりサーバの具体的な存在や構成は利用者に意識させることはなく，資源管理・運用者が責任をもつ方法である．

このようなコンピューティングモデルを実現するには，一般的に提供者は複数のサーバを管理・運用することになる．管理・運用の側からすると，人件費ならびに運用コストを低くする必要があるため，例えばトランザクション処理が日中のピークを過ぎ，夜間などの閑散となった時間帯では，複数のサーバごとに分散処理しているサービス機能を限定されたサーバに集約できれば，サーバ運用の人件費ならびに電気料金などの節約になる．このような要求に応えるには，あるサーバ機能を別のサーバに集約する機能が必要となる．集約の対象となるサーバは一般的には，OS とアプリケーションが組み合わされているので，そのまま集約されるサーバ上で動作しなければならない．したがって，集約されるサーバを仮想計算機構成にしておけば，この集約運転の可能性がある．

■7 群 － 3 編 － 9 章

9-3 演習問題

（執筆者：吉澤康文）［2013 年 2 月 受領］

(1) マルチパーソナリティを実現する方法を 2 通り述べよ.
(2) 上記の各方法について，その利点と欠点を述べよ.
(3) 仮想計算機方式の欠点を解決する方法は何か述べよ.
(4) マイクロカーネルの本来の目的は何か.
(5) 単一構成のカーネル構成法の利点と欠点を述べよ.
(6) UNIX が単階層のカーネルである利点をあげよ.
(7) ファイルシステムはカーネルである必要があるか．カーネルした場合の利点，欠点などを考察せよ.
(8) モノリシックとマイクロカーネルの構成方式の特徴を述べ，相互の利点，欠点を述べよ.