(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号
特開2001－318797
（P2001－318797A）
(43)公開日　平成13年11月16日(2001.11.16)

(51)Int.Cl.[7]		識別記号	FI			テーマコード(参考)
G06F	9/46	350	G06F	9/46	350	5B017
	12/14	310		12/14	310L	5B085
	15/00	330		15/00	330A	5B098

審査請求　有　　請求項の数42　ＯＬ　（全 23 頁）

(21)出願番号　特願2000－281632(P2000－281632)

(22)出願日　平成12年９月18日(2000.9.18)

(31)優先権主張番号　０９／５６８６２９

(32)優先日　平成12年５月10日(2000.5.10)

(33)優先権主張国　米国（ＵＳ）

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

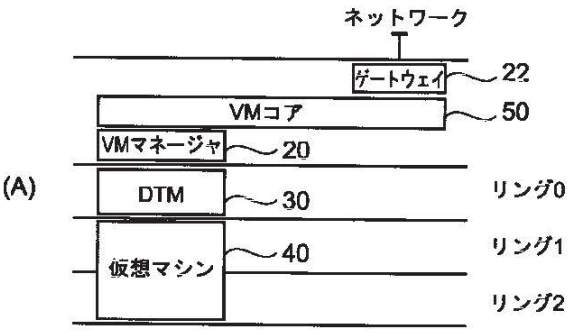
(72)発明者　レスリー・ジェイ・フレンチ
アメリカ合衆国、ニュージャージー　08540　プリンストン，４　インディペンデンス　ウエイ、エヌ・イー・シー・ユー・エス・エー・インク内

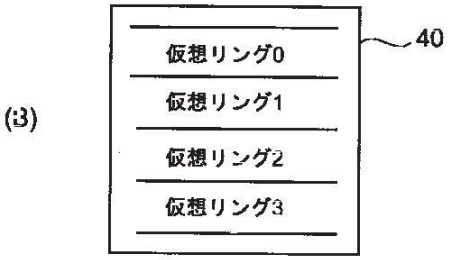
(74)代理人　100097157
弁理士　桂木　雄二

Ｆターム(参考)　5B017 AA01 BA03 BB06 CA16
5B085 AE00 AE06 AE29
5B098 GD14 GD21 HH01

(54)【発明の名称】　自動データ処理装置

(57)【要約】
【課題】　コードがプロセッサ上で直接に実行される仮想環境を提供し、各仮想環境の周りにセキュリティシェルを実装した、仮想マシンアーキテクチャを実現する。
【解決手段】　仮想環境内の特権リング階層が、ハードウェアにより利用される特権リング階層のサブセットから提供される。仮想環境内からは、最上位特権を要求するハードウェアへのアクセスは厳格に制御される。仮想マシンと外界との間の通信は、シミュレートされたネットワークを通じて行われる。ネットワークは、ファイアウォールとして作用する仮想マシンによって保護され、暗号化を用いてデータトランザクションが保護される。ハードウェアは、仮想環境には利用可能でない暗号化鍵を保持する。仮想環境は、ハードウェア鍵を利用した安全なトランザクションで外部サーバから取得可能な固有の鍵を受信する。





【特許請求の範囲】

【請求項１】　システムリソースおよびメモリへのアクセスを制御するために少なくとも３個の実特権リングを用いて情報を処理するとともに、少なくとも１つの仮想マシン環境を提供し、情報および特権は所定のリングに関連づけられている自動データ処理装置において、

実リングプロパティを変更する処理が制限された最上位特権の実リングと、第２位特権の実リングと、残りの実リングと、からなる実環境と、

前記最上位特権の実リングで動作する仮想マシンマネージャと、

前記残りの実リングおよびメモリの一部をからなる仮想マシン環境と、

前記第２位特権の実リングで動作し、前記仮想マシン環境を出入りする情報転送を処理する第１データ転送マネージャと、

前記仮想マシン環境内にある階層仮想特権リングと、

前記仮想マシン環境内の情報に対する特権リングの関連づけを含み、前記仮想マシン環境内に形成された仮想環境ディスクリプタテーブルと、からなり、前記仮想マシン環境内の情報は前記階層仮想特権リングに関連づけられ、前記メモリの一部内に記憶され、

前記自動データ処理装置によって処理される情報は、同位または下位の特権の実リングに関連づけられたシステムリソースおよび情報にアクセス可能であり、

前記仮想マシン環境内の情報は、前記自動データ処理装置によって処理される場合、同位または下位の特権の仮想特権リングに関連づけられたシステムリソースおよび情報にアクセス可能であるとともに、前記仮想マシン環境の前記メモリの一部内のメモリロケーションにのみアクセス可能であり、

仮想特権リングに関連づけられた情報にとって利用可能な実特権は、前記自動データ処理装置によって処理される場合、前記残りの実リングのうち前記仮想マシン環境を含む実リングを超える実特権へのアクセスが要求されるときに前記仮想マシンマネージャによって制御され、

前記メモリの一部は主メモリおよび補助記憶領域を含む、ことを特徴とする自動データ処理装置。

【請求項２】　前記自動データ処理装置は、

仮想特権リングに対して前記仮想マシンマネージャによって生成され、前記仮想マシン環境内の情報を実特権リングの階層にマップするシャドウディスクリプタテーブルをさらに有し、

前記仮想環境ディスクリプタテーブルは、前記仮想特権リングの階層への関連づけを含み、

仮想特権リングに関連づけられた情報が前記自動データ処理装置によって処理されるとき、該情報が関連づけられた仮想特権リングに対応するシャドウディスクリプタテーブルが、アクセス特権を決定するために使用される、ことを特徴とする請求項１記載の自動データ処理装置。

【請求項３】　前記仮想マシンマネージャは、仮想特権リング内で処理を開始するときに、処理されるべき情報に対応する仮想特権リングに対するシャドウディスクリプタテーブルを生成する、ことを特徴とする請求項２記載の自動データ処理装置。

【請求項４】　前記仮想マシンマネージャは、前記仮想環境ディスクリプタテーブルに変更が生じたときにシャドウディスクリプタテーブルを更新することを特徴とする請求項３記載の自動データ処理装置。

【請求項５】　前記自動データ処理装置によって処理される情報が仮想特権リングを変更するときに、前記仮想マシンマネージャは、処理の変更先の仮想特権リングに対するシャドウディスクリプタテーブルを生成することを特徴とする請求項３記載の自動データ処理装置。

【請求項６】　前記仮想マシンマネージャは、情報が処理された各仮想特権リングに対するシャドウディスクリプタテーブルを含む複数のシャドウディスクリプタテーブルを保持することを特徴とする請求項５記載の自動データ処理装置。

【請求項７】　前記仮想マシンマネージャは、前記仮想環境ディスクリプタテーブルに変更がなされるときに、前記複数のシャドウディスクリプタテーブルを更新することを特徴とする請求項６記載の自動データ処理装置。

【請求項８】　前記仮想マシンマネージャは、前記メモリの一部内の情報が再配置されるときに、前記複数のシャドウディスクリプタテーブルを再生成することを特徴とする請求項６記載の自動データ処理装置。

【請求項９】　前記仮想マシンマネージャは、前記仮想特権リングの階層の各仮想リングに対するシャドウディスクリプタテーブルを含む複数のシャドウディスクリプタテーブルを生成し保持することを特徴とする請求項２記載の自動データ処理装置。

【請求項１０】　前記仮想マシンマネージャは、前記仮想環境ディスクリプタテーブルに変更が生じたときに前記複数のシャドウディスクリプタテーブルを更新することを特徴とする請求項９記載の自動データ処理装置。

【請求項１１】　前記仮想マシンマネージャは、前記メモリの一部内の情報が再配置されるときに、前記複数のシャドウディスクリプタテーブルを再生成することを特徴とする請求項９記載の自動データ処理装置。

【請求項１２】　実特権リングの階層は４個のリングを有し、

第３位特権の実リングは、最上位特権の仮想リングを有し、

第４位特権の実リングは、仮想特権リングの階層の残りの部分を有することを特徴とする請求項２記載の自動データ処理装置。

【請求項１３】　最上位特権の仮想リングに対して前記仮想マシンマネージャによって生成されるシャドウディ

スクリプタテーブルは、前記仮想マシン環境内のすべての情報から前記第３位特権の実リングへのマッピングからなり、
前記仮想リングの階層の残りの部分の仮想リングに対して前記仮想マシンマネージャによって生成されるシャドウディスクリプタテーブルは、
前記最上位特権の仮想リングに関連づけられた情報に対する、前記第３位特権の実リングへのマッピングと、
前記シャドウディスクリプタテーブルが生成された仮想特権リングと同位または下位の特権の仮想リングに関連づけられた情報に対する、前記第４位特権の実リングへのマッピングと、
前記シャドウディスクリプタテーブルが生成された仮想特権リングより上位の特権の仮想リングに関連づけられた情報に対する、前記最上位特権の実リングへのマッピングと、からなり、それにより、前記仮想マシンマネージャは、前記仮想リングの階層の残りの部分内の上位特権の仮想リングに関連づけられた情報へのアクセスを求める要求をトラップする、ことを特徴とする請求項１２記載の自動データ処理装置。
【請求項１４】　実特権リングの階層への情報の関連づけからなり、シャドウディスクリプタテーブルを含み、かつ前記実環境の全体に対するグローバルディスクリプタテーブルをさらに有することを特徴とする請求項２記載の自動データ処理装置。
【請求項１５】　前記仮想マシン環境内の情報は、
最上位特権を有する仮想特権リングと、最下位特権を有する仮想特権リングと、からなる２個のリングのみに関連づけられる、ことを特徴とする請求項１記載の自動データ処理装置。
【請求項１６】　実特権リングの階層は４個の特権リングを有することにより、前記仮想マシン環境は２個の実リングを有し、
第３位特権の実リングは、最上位特権の仮想リングを有し、
第４位特権の実リングは、最下位特権の仮想リングを有することを特徴とする請求項１５記載の自動データ処理装置。
【請求項１７】　前記仮想マシン環境内から処理される情報は、前記仮想環境ディスクリプタテーブル内の前記仮想特権リングの階層への関連づけを記録しようと試み、
前記仮想マシンマネージャは、前記仮想特権リングの階層への関連づけの代わりに、前記仮想環境ディスクリプタテーブル内の実特権リングの階層への関連づけを記録し、
前記仮想マシン環境内に関連づけられた情報が前記自動データ処理装置によって処理されるとき、前記仮想環境ディスクリプタテーブルが、アクセス特権を決定するために使用される。ことを特徴とする請求項１記載の自動データ処理装置。
【請求項１８】　前記仮想マシンマネージャは、前記仮想環境ディスクリプタテーブルが形成されるとき、および、前記仮想環境ディスクリプタテーブルを修正しようとする試みがなされるごとに、前記仮想環境ディスクリプタテーブル内の代わりの関連づけを記録する、ことを特徴とする請求項１７記載の自動データ処理装置。
【請求項１９】　前記仮想マシンマネージャは、前記仮想環境ディスクリプタテーブル内の仮想特権リング関連づけを記録しようとする試みを許可した後、前記仮想マシンマネージャは、上書きにより、実特権リング関連づけを置き換えることを特徴とする請求項１７記載の自動データ処理装置。
【請求項２０】　前記仮想環境ディスクリプタテーブル内の仮想特権リング関連づけを記録しようとする試みは、前記仮想マシンマネージャによってトラップされ、前記仮想マシンマネージャは、代わりに実特権リング関連づけを記録することを特徴とする請求項１７記載の自動データ処理装置。
【請求項２１】　前記仮想マシン環境と別の仮想マシン環境との間、および、前記仮想マシン環境と前記実環境との間の情報転送は、データ転送マネージャによって処理されることを特徴とする請求項１記載の自動データ処理装置。
【請求項２２】　前記自動データ処理装置は、データ転送マネージャスプールをさらに有し、
データ転送マネージャによって処理される情報転送は前記データ転送マネージャスプールに一時的に格納されることを特徴とする請求項１記載の自動データ処理装置。
【請求項２３】　前記データ転送マネージャは、仮想マシン環境を出る情報転送を確認することをユーザに求めることを特徴とする請求項２２記載の自動データ処理装置。
【請求項２４】　前記第１データ転送マネージャおよび前記仮想マシン環境を含む仮想ファイアウォールをさらに有することを特徴とする請求項１記載の自動データ処理装置。
【請求項２５】　前記自動データ処理装置は、
前記第２特権の実リングで動作し、仮想マシン環境を出入りする情報転送を処理する第２データ転送マネージャをさらに有し、
前記仮想ファイアウォールは、前記第２データ転送マネージャをさらに有し、
前記第１データ転送マネージャは、前記仮想マシン環境と、ファイアウォールの外部との間の情報転送を処理し、
前記第２データ転送マネージャは、前記仮想マシン環境と、ファイアウォールの内部との間の情報転送を処理する、
ことを特徴とする請求項２４記載の自動データ処理装

置。
【請求項２６】　前記第１データ転送マネージャは、前記仮想マシン環境と、ファイアウォールの外側との間の情報転送を、前記仮想マシン環境と、ファイアウォールの内側との間の情報転送から隔離することを特徴とする請求項２４記載の自動データ処理装置。
【請求項２７】　前記自動データ処理装置は、ファイアウォールスプールをさらに有し、
前記ファイアウォールを通しての情報転送は前記ファイアウォールスプールに一時的に格納されることを特徴とする請求項２４記載の自動データ処理装置。
【請求項２８】　前記ファイアウォールは、ファイアウォールの内側からファイアウォールの外側への情報転送を確認することをユーザに求めることを特徴とする請求項２７記載の自動データ処理装置。
【請求項２９】　前記自動データ処理装置は、実環境秘密暗号化鍵を記憶する不揮発性メモリをさらに有し、
前記実環境秘密暗号化鍵は、前記自動データ処理装置に固有であり、
前記実環境秘密暗号化鍵へのアクセスは、前記最上位特権の実リングおよび前記第２位特権の実リングに制限されることを特徴とする請求項１記載の自動データ処理装置。
【請求項３０】　前記仮想マシンマネージャは、前記実環境秘密暗号化鍵に対応する実環境公開暗号化鍵を用いて、前記仮想マシン環境から前記メモリの一部の前記補助記憶領域への情報転送を暗号化し、
前記仮想マシンマネージャは、前記実環境秘密暗号化鍵を用いて、前記メモリの一部の前記補助記憶領域から前記仮想マシン環境への情報転送を復号し、
前記暗号化および復号は、前記仮想マシン環境内からは不可視であることを特徴とする請求項２９記載の自動データ処理装置。
【請求項３１】　実環境暗号化鍵は、前記仮想マシン環境を出る情報転送を暗号化し、前記仮想マシン環境に入る情報転送を復号するために、前記仮想マシン環境の外部で適用され、
暗号化および復号の適用は、前記仮想マシン環境内からは不可視であることを特徴とする請求項２９記載の自動データ処理装置。
【請求項３２】　前記仮想マシン環境内に転送される復号された情報は、他の仮想マシン環境にはアクセス不可能であることを特徴とする請求項３１記載の自動データ処理装置。
【請求項３３】　実環境暗号化鍵は、前記仮想マシンマネージャによって適用されることを特徴とする請求項３１記載の自動データ処理装置。
【請求項３４】　実環境暗号化鍵は、前記データ転送マネージャによって適用されることを特徴とする請求項３１記載の自動データ処理装置。
【請求項３５】　前記仮想マシン環境には、仮想環境秘密暗号化鍵および仮想環境公開暗号化鍵が割り当てられ、これらの暗号化鍵は、実環境および他の仮想マシン環境の暗号化鍵とは異なり、他の仮想マシン環境にはアクセス不可能であることを特徴とする請求項２９記載の自動データ処理装置。
【請求項３６】　仮想環境暗号化鍵は、前記仮想マシン環境を出る情報転送を暗号化し、前記仮想マシン環境に入る情報転送を復号するために、前記仮想マシン環境の外部で適用され、
前記仮想マシン環境内に転送される復号された情報は、他の仮想マシン環境にはアクセス不可能であることを特徴とする請求項３５記載の自動データ処理装置。
【請求項３７】　仮想環境暗号化鍵は、前記仮想マシンマネージャによって適用されることを特徴とする請求項３６に記載の装置。
【請求項３８】　仮想環境暗号化鍵は、前記データ転送マネージャによって適用されることを特徴とする請求項３６記載の自動データ処理装置。
【請求項３９】　仮想環境暗号化鍵は、前記仮想環境内にアクセス可能であることにより、前記仮想環境内のプロシージャは、前記仮想マシン環境に入る情報転送を復号し、前記仮想マシン環境を出る情報転送を暗号化することを特徴とする請求項３５記載の自動データ処理装置。
【請求項４０】　前記自動データ処理装置は、外部鍵サーバから仮想環境暗号化鍵を取得することを特徴とする請求項３５記載の自動データ処理装置。
【請求項４１】　前記自動データ処理装置と外部鍵サーバとの間の、鍵取得に関するトランザクションは、実環境秘密暗号化鍵を用いて安全にされることを特徴とする請求項４０記載の自動データ処理装置。
【請求項４２】　前記仮想特権リングの階層の数は、前記実特権リングの階層の数に等しいことを特徴とする請求項１記載の自動データ処理装置。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は階層的保護リングを利用してシステムオペレーションを規制するディジタルデータ処理システムの分野に係り、特に、１以上の仮想マシン（ＶＭ：virtual machine）を備えるとともに、メモリ内のロケーションへのアクセスおよび一部の命令の実行可能性を規制する階層構成された３個以上の保護リングを有し、仮想マシンの外部にアクティブセキュリティ層を備えたコンピュータシステムに関する。
【０００２】
【従来の技術】「パーソナル」コンピューティングデバイス（ＰＣ）のセキュリティ（安全性）に関する問題は、多くのコンピューティングの領域を通じてますます重要になっている。ＰＣはデスクトップマシンの主流に

なりつつあり、同時にスタンドアローン型コンピュータとネットワークマシンとの間の境界線は曖昧になっている。従来のワークステーションは、小型メインフレームとともに、ＰＣに取って代わられているが、ＰＣは安全性が低いことでも知られている。オープン（開放型）アクセスネットワーク（例えば、インターネットや、ケーブルモデムプロバイダのアクセスネットワーク）にますます多くのシステムが接続されることにより、これらのシステムは、さまざまなアクセスレベルでの侵入を受けやすくなっている。モデムデータトランザクションのうちの最も単純なものでもそれに関わる情報は大量であり、また、ネットワーク駆動型アプリケーションが急増しているため、システムセキュリティの最初で最後の防衛線はいつも「信頼」ということになってしまう。

【０００３】セキュリティの問題に対するパッチワーク的な解決法がいくつか提案され、あるいは実装されている。最も単純なアプローチの１つは、コンピュータがリムーバブルメディア（通常はフロッピー（登録商標）ディスク）からブートしないようにすることにより、「ブートブロック」ウィルスの拡散を防ぐことである。この方法では、ユーザがファームウェア基本入出力システム（ＢＩＯＳ）にアクセスし（これはパスワード保護されていることが多い）、フロッピーからブートすることができる前にマシンをリブートすることが必要である。

【０００４】第２のアプローチは、システム内のすべての既存のソフトウェアを、より安全なバージョンで置き換えることである。しかし、ソフトウェアアプリケーションの複雑さは増大しているため、悪意のあるプログラマが利用しようとする（セキュリティ）ホールやバグの供給は尽きることがない。

【０００５】第３のアプローチは、パスワードの使用および記憶データの暗号化による「秘密のベール」である。しかし、秘密性をセキュリティと同視するのは見当違いである。データを秘匿することは、発見を困難にするかもしれないが、問題の解決にはならない。データおよびパスワードファイルを窃取した後に力ずくで解読する方法は別としても、秘密化方式は、「内部」からの攻撃に非常に弱い。このような攻撃の一例は、偽装プロセスが、ユーザの鍵（キー）入力をモニタすることによって、コンピュータの暗号化鍵のコピー、または、ユーザのパスワードのコピーを取得するというものである。その後、偽装プロセスは情報を外界にブロードキャストし、侵入者が、最も高度な暗号化技術でさえもバイパスすることを可能にしてしまう。

【０００６】第４のアプローチは、ソフトウェアまたはハードウェアの「ファイアウォール」を使用し、データダウンロードおよびネットワークトランザクションをモニタするというものである。しかし、ファイアウォールは、そのフィルタと同程度にしか有用ではなく、情報の通過を許可するかどうかを決定するのに信頼に頼らなければならないことが多い。

【０００７】第５の方法は、ウィルスのコードや攻撃に対して、コードをスキャンしソフトウェア実行をモニタすることである。この方法はＰＣで広まっているが、その弱点も多く文書化されている。特に、未認知のコードがオペレーティングシステムの欠陥から侵入し、アンチウィルスモニタより下の動作レベルにあるシステムリソースへのアクセスが可能となるときには、この方法は弱い。この方法は、ワードプロセッサのマクロやネットワークブラウザアプリケーションのような、アプリケーション環境内で実行するためのコードにとっては、特に弱い。必然的に、マシンを完全に孤立させたままにするという「軍用型」アプローチを実装するのでなければ、未検出の侵入による内部からの攻撃は、このようなセキュリティ方式を危うくする。

【０００８】第６のアプローチは、「軍用型」アプローチと同様に、エミュレートされた「サンドボックス」環境内でソフトウェアを実行することである。ソフトウェアは、コンピュータシステムエミュレータ内にロードされて実行され、疑いのある活動がモニタされることにより、汚染された可能性のあるアプリケーションに、より大きいシステムがさらされる危険が最小化または除去される。しかし、多くのウィルスは、特定のイベントが発生するまで眠っているため、安全性を確実にするには、疑わしいアプリケーションを無期限に隔離する必要がある。そうでなければ、誤ってウィルスがないとみなされた汚染したアプリケーションがサンドボックスの外に出て、そのアプリケーションが許可されたマシン「全体」に感染する。しかし、アプリケーションを環境エミュレータ内で無期限に実行することは、長期間の動作にとっては望ましくない。エミュレータは、コードの実行にかなりの複雑さを追加する。サンドボックス環境内のソフトウェアは、コンピュータのプロセッサによって直接に実行されるのではなく、命令ごとにシミュレートされることにより、システム動作が遅くなるからである。

【０００９】さらに、一部の解決法では、サンドボックス環境を従来のオペレーティングシステムのフレームワーク内に作成しようとしている。このような実装は、それが実行される基礎となるオペレーティングシステムと同程度にしか安全ではない。例えば、サンドボックスと同じコンピュータ上で実行される特権プロセスは、サンドボックス環境のデータおよびメソッドにアクセスすることが可能である。このような特権プロセスは、サンドボックス環境の制御下にないコンピュータの一部で実行されている悪意のあるアプリケーションの結果である可能性があり、あるいは、サンドボックス実装における不測の抜け穴から生じた可能性がある。

【００１０】セキュリティに対するこれらのさまざまな「２レベル」モデルは、「全か無か」の保護であり、検証コードがオペレーティングシステムと高い特権レベル

を共有するという欠点を有する。そのコードが正しく動作することをどのくらい信頼することができるかが問題となる。インタフェースにおける小さい穴（ホール）の存在でさえ、保護方式の全体を破壊する可能性がある。

【００１１】レガシー（従来の）オペレーティングシステムの主な問題点のうちの１つは、「意図」(intention)を判断することにある。セキュリティモニタが、あるオペレーションが意図されたものかどうかを推論することは困難である。ディスクフォーマットユーティリティは、ウィルスとは区別できない方法で、デバイスのブートブロックに書き込みをしようとする。ユーザがメーリングリストのすべてのユーザに電子メールを送るのは、ネットワークを通じて拡散するメールワームとは区別できないことがある。どれが正当なオペレーションであるかを「後知恵」で判断しようとすることは、最終的に失敗しやすい。ハッカーは、発見的なプロセスで、自分のコードが本当に真正であると確信させることがうまくなっているからである。

【００１２】これらのセキュリティ方法の下の、ハードウェア・ファームウェアのレベルでは、ＰＣ（例えば、Intelのアーキテクチャに基づくもの）は、特権リングとセグメントテーブルの階層を利用して、プロセスどうしが干渉や妨害をしあわないようにしている。

【００１３】メモリ４ａ（図１のＡ）は、セグメント６ａ～６ｅと呼ばれるブロックに分けられる。あるプロセスから個々のセグメントが見えるかどうか（可視性）と、そのプロセスに与えられるアクセス権は、各プロセスに割り当てられるセグメントテーブルによって決定される。ひとまとめにして、プロセスの権利は、１つ以上のディスクリプタテーブル（ＤＴ：descriptor table）８によって決定される。メモリセグメント６ａ～６ｅに関連づけられたディスクリプタ１０ａ～１０ｅ（図１のＢ）は、セグメントの開始アドレスおよび大きさ（長さ）、ならびに、その領域に対するアクセス権を指定する。

【００１４】セグメントは、プロセッサのアーキテクチャにより、複数のプロセスどうしの間で共有されることも可能である。あるセグメントが同一のアクセス権ですべてのプロセスから見えるようにされる場合、セグメントはグローバルテーブルにおかれることが可能である。さらに、各プロセスは、そのプロセスのみに関連づけられたディスクリプタを保持するローカルテーブルを有することも可能である。このようにして、同じメモリ領域が、相異なるプロセスからの相異なるアクセス権を有することが可能となる。

【００１５】例えばIntelアーキテクチャのＣＰＵを用いたコンピュータでは、メインメモリに配置された単一のグローバルディスクリプタテーブル（ＧＤＴ：Global Descriptor Table）があり、これは、アーキテクチャ固有のグローバルディスクリプタテーブルレジスタ（ＧＤＴＲ：Global Descriptor Table Register）を通じてアクセスされる。このアーキテクチャでは、複数のＧＤＴを設け、プログラム実行中に動的にアクティブにすることが可能である。さらに、各プロセスはローカルディスクリプタテーブル（ＬＤＴ：Local Descriptor Table）を有することが可能であり、これは、上記と同様にして設けられたローカルディスクリプタテーブルレジスタ（ＬＤＴＲ：Local Descriptor Table Register）を通じてアクセスされる。

【００１６】このプロセッサアーキテクチャは、外部ソースからまたはソフトウェアによって生成される割込みに応答してのプロセッサの動作を決定するために用いられる、もう１つのテーブルを備える。このテーブルは、割込みディスクリプタテーブルレジスタ（ＩＤＴＲ：Interrupt Descriptor Table Register）を通じてアクセスされる。

【００１７】ハードウェアは、ディスクリプタテーブルを参照することによって、プロセスにより使用されるアドレスをチェックし、アドレスがそのプロセスに割り当てられた空間の外部にある場合、例外が発生する。しかし、いくつかのセグメントは、いくつかのプロセスによって共有されている。これらの共有セグメントを保護するため、図２に示すような、特権リングの階層を設ける。各ディスクリプタテーブルエントリは、テーブル内のそのエントリに対するアクセス権を決定するために使用されるリング保護レベルに関連づけられる。実行中のプロセスの一部として実行されるプロシージャを含めて、すべての情報は１つのある特権リングに割り当てられる。プロシージャのカレント実行リングは、プロセスの実行中に変化することがあり、さらに、そのプロシージャに関連づけられたメモリの可視性を決定する。

【００１８】４個のリング０～３の階層において、最も内側のリング０は最上位の特権のリングであり、最も外側のリング３は最下位の特権のリングである。内側のリングにあるプロシージャは、外側のリングにあるデータにアクセスすることができる（矢印１２）。そのため、最も内側のリングで動作するプロシージャ（例えば、オペレーティングシステムのプロシージャ）は、すべてのリングにあるデータにアクセスすることができる。

【００１９】逆に、外側のリングにあるプロシージャは、保護違反例外を引き起こさずには、内側のリングにあるデータにアクセスすることはできない（矢印１４）。内側リングにあるデータにアクセスするためには、プロシージャは「コールゲート」１８を通じて、そのデータを含むリングに分岐しなければならない。コールゲートとは、ハードウェアによって認識される特殊な「プロシージャコール」ディスクリプタである。いったんゲートを通ると、上位に分岐することを許されたプロシージャが下位のリングに戻る保証はない。また、プロシージャが目標のリングに達した後、そのプロシージャ

が害悪を働かないという保証もない。

【００２０】仮想マシン環境は、システムセキュリティを維持するための有用なツールである。上記のエミュレートされたサンドボックスはその一例である。１つ以上の仮想マシンを備えたコンピュータシステムは通常、リングの階層を、より少数の実リング（単一のリングのこともある）に圧縮している。実リングには仮想マシン環境が存在し、仮想マシン内から命令が実行されるときに、この人工的環境を受け持つソフトウェアに、特権を決定することをまかせている。例えば、仮想マシンがリング３内でエミュレートされるとき、仮想マシン内のすべてのリングはリング３内からシミュレートされる。

【００２１】仮想マシン内からのコードをプロセッサ上で直接に実行することによって、エミュレータの実行が遅いという問題点を克服する、別の仮想マシン環境が開発されている。各仮想マシン環境内からのコードは、通常のタスクと同様にして実行される。環境管理ソフトウェアは、既存のオペレーティングシステム内の、または、オペレーティングシステムと連携して、オペレーティングシステム内のデバイスドライバを利用し、システムリソースへのアクセスを規制するプロセスとして動作する。残念ながら、これらの方法は、環境シミュレーションの速度の欠点は克服するものの、安全でないレガシー環境に埋め込まれたサンドボックスは、コード実行をシミュレートするソフトウェアベースのエミュレータのセキュリティ保護に欠けている。

【００２２】

【発明が解決しようとする課題】これらはすべて、過度のエミュレーションを避けながら、インタフェースにおけるホール（穴）を塞ぐ環境についてのものである。環境は、修正なしのレガシーオペレーティングシステムの使用が可能になるように、オープンでしかもサンドボックス的でなければならず、同時に、信頼性の高い多層保護および防御可能なセキュリティ制限を提供しなければならない。

【００２３】

【課題を解決するための手段】上記の問題点を解決するため、本発明の目的は、コードがプロセッサ上で直接に実行されるが、各仮想環境の周りにセキュリティシェルを実装した、仮想マシンアーキテクチャを提供することである。シェルを通じてクリティカルなシステムリソースにアクセスすること（ネットワークアクセスを含む）は、制限的であるが透過的に制御される。セキュリティ制限は、各「サンドボックス」の完全性を維持しながら、ネットワークおよびシステムのリソースへの見かけ上オープンなアクセスを可能にする。

【００２４】本発明は、すべてのインタラクションを検証しようとするのではなく、データが通る非常に狭く定義されたインタフェースを有する仮想マシンを通じて、孤立した環境を提供する。この実装は、仮想マシン内で動作するゲストレガシーオペレーティングシステムからは見えない。

【００２５】データ検証メカニズムは、同じハードウェア内に存在する孤立した仮想環境どうしの間での転送に適用される。セキュリティメカニズムはモジュール的であり、仮想ファイアウォール、ハードウェア間およびハードウェア内暗号化、ならびに、ブートブロック検証を含む。「安全でない」環境に物理ディスクを移動し、オペレーティングシステムコンフィグレーションを変更し、「安全な」環境にそのディスクを戻すことに対する保護が設けられる。廃棄または窃取されたドライブからのデータ窃取は、ディスクブートブロックの破壊とともに、防止される。不揮発性ＲＡＭデータは、制限されたアクセスによって隠され、あるいは、保護されることが可能である。各オペレーティング環境は孤立している。ファイアウォールが、マシンデータとネットワークの間に作成され、機密データを含む安全な環境から「高リスク」アプリケーションを隔離する。

【００２６】本発明は、好ましくは、物理リソースを保護する小型のコアオペレーティングシステム（例えば、１０００行のオーダーのコード）であり、それにより、マシンパフォーマンスの損失を最小にし、オペレーティングシステムの全部または一部をＲＯＭのような保護されたブート可能メモリに記憶することを可能にする。コアを小さくすることにより、将来変化するセキュリティモデルに従っていることを示すために正当性証明を適用すること、および、既存のシステムへの影響を最小にしてマシンパフォーマンスの損失を防ぐことが可能となる。

【００２７】仮想マシンコアオペレーティングシステムは、特権アプリケーションとして動作する仮想マシンマネージャと同様、最上位特権リング内で動作する。仮想マシンマネージャは、各仮想環境のためのシステムリソースのエミュレーションを管理する。データ転送マネージャ（ＤＴＭ：data transfer manager）は、２番目に高い特権リング内で動作し、各仮想環境を出入りするデータ転送を管理する。仮想環境内には、通常のオペレーティング環境をシミュレートするために仮想特権リングが作成される。各仮想マシンは自分自身のオペレーティングシステムを有するが、マシンリソースへのすべてのアクセスは仮想マシンコアを通して検証されるという制限された環境にある。

【００２８】一般に、仮想マシンにおけるほとんどの命令は「ネイティブ」に実行されるが、リング０実行レベルを必要とする命令のみはコアにトラップされる。実装の一部は、仮想マシンのための特権オペレーションを隠すデータ構造体（例えば、プロセッサステータスレジスタ）の管理である。この方式では、レガシーオペレーティングシステムには、自分自身の特権レベルの変化を見かけ上追跡するシステムレジスタのセットが見えること

になるが、リング０命令の使用、物理メモリへのアクセスあるいは物理デバイスへのアクセスをしようとすると、まず仮想マシンコアを通ることになる。

【００２９】修正されていないオペレーティングシステムを仮想環境で実行するため、ゲストシステムには、ネイティブマシンと同一であるように見えるメモリモデルが提示される。例えば、各マシンには、（仮想）リセット時にロケーションＦＦＦＦ：ＦＦＦ０に（おそらくは相異なる）ブートＲＯＭが見えなければならない。そのため、仮想メモリ管理システムは、実マシン上の相異なる物理アドレスに同じ論理アドレスをマップする。

【００３０】各マシンが自分自身のブートシーケンスを通して動作することを保証するため、相異なる仮想マシンに返される結果は実際のハードウェアを反映しないことがある。例えば、マシンは、ハードウェアにあるよりも多いまたは少ない物理メモリにアクセス可能であることがあり、また、プローブに応答するバス上のすべてのデバイスが見えるとは限らない。各仮想環境に個別化されたブートＲＯＭを設けることにより、ＢＩＯＳデータ構造体が、その仮想マシンに必要な仮想状態を反映するようにされる。

【００３１】同様に、ゲストマシンは、ハードウェアリソースにアクセスすることを阻止される。入出力命令、および、メモリマップドページを通じて外部デバイスにアクセスする命令もまた、コアにトラップされる。これを実現するため、コアは、可能な場合、与えられたタスクに対して個々の入出力空間アドレスがロックアウトされることが可能であるようなメモリ管理システムを使用する。

【００３２】ハードウェア割込みは、コアハンドラにトラップされ、そこから適当なマシンにディスパッチされる。このメカニズムは、１つの仮想マシンしかない場合でも適用しなければならない。割込みをイネーブル（有効）またはディスエーブル（無効）にする命令は、仮想環境内でエミュレートされているだけだからである。仮想マシンのリング０にデバイスドライバをロードすることが許可される場合に限り、割込みを直接にそのハンドラにディスパッチすることができる。これが当てはまる唯一の場合は、パフォーマンスが問題となる場合であろう。例えば、自分自身の検証領域で実行されている大容量デバイス（例えばイーサネット（登録商標）コントローラ）に対する複数回のデータコピーやマシン再スケジュールを避けるためである。明らかに、このメカニズムは、ファイアウォールのような高度に信頼されたアプリケーションに対してのみ利用可能である。

【００３３】仮想マシンの概念は、ハード（ディスク）ドライブのような、マシン上の補助メモリ（補助記憶領域）にも拡張される。マシンどうしが同じ物理デバイスを共有していても、それらが専用のドライブを有しているかのように、絶対ブロックアドレスへのアクセスは、トラップされ、仮想ディスクアドレスに変換される。

【００３４】ディスクへの物理攻撃を考慮して、すべてのアクセスをトラップすることにより、本発明のアーキテクチャ内で他の付加価値サービスが可能となる。例えば、本発明は、透過的なディスクミラーリング、あるいはＲＡＩＤ(Redundant Arrayof Independent Disks)システムを実装して、故障したハードウェアからのデータ回復を改善することができる。また、このレベルで可能なこととして、コアは、不揮発性ＲＡＭに記憶された鍵を用いて仮想マシン上のすべてのデータを暗号化し、ディスクが他のマシン上では読むことができないようにすることができる。さらに完全な保護のために、コアシステムをマシンごとの不揮発性ＲＡＭ鍵によって保護し、仮想マシンデータが、マシン鍵とユーザによる鍵入力の両方からなる２レベル鍵を使用することが可能である。

【００３５】これらの方式を使用することを選択すると、システムセキュリティと全体としてのシステムパフォーマンスへの影響との間の妥協を必要とする。暗号化方法は、コンポーネント障害時にコンポーネントを「ホットスワップ」することがほとんど不可能であるという欠点を有する。また、ディスククラッシュからのデータ回復がずっと困難になる。

【００３６】可能性として、ネイティブシステム上では別個のリング（１～３）で実行されるコードが、ゲストシステムでは同じ物理リング内で実行される心配がある。これにより、ユーザレベルのアプリケーションが、アクセスすることができないはずのデータ構造体を変更することができてしまう可能性がある。この心配は、メモリ管理サブシステムを用いた仮想リングシステムを実装することにより解決される。このようにして、異なる仮想リングのページは別個のまま保持され、同じ物理リング内のタスクどうしの間のタスク切り替えは、メモリアクセス制御をも切り替える。

【００３７】本発明の基本的な実装は、

・好ましくは安全なブートＲＯＭに記憶された、仮想マシンコアおよび仮想マシンマネージャのためのローダ、

・コンフィグレーションデータおよびセキュリティ鍵を保持するための不揮発性ＲＡＭ内の領域、

・個々の仮想マシンを実行することを担当する仮想マシンコアスケジューラ、

・ゲスト仮想マシンのコンフィグレーション（設定）、ロード、およびセキュリティポリシーを処理する仮想マシンマネージャ、

・ゲスト仮想マシンのコンフィグレーション仕様、

・ゲスト仮想マシンのための仮想ブートＲＯＭ、

・ゲスト仮想マシンのための仮想不揮発性ＲＡＭパラメータ領域、

・ゲスト仮想マシンのためのリング１タスクにロードされる安全なアプリケーション、

・ゲスト仮想マシンのためのファイルシステムを含むデ

ィスクパーティション（オペレーティングシステムおよびアプリケーションコードを含む）
を有する。
【００３８】ほとんどのマシンは、さまざまなハードウェアを含む。例えば、リムーバブルメディアデバイス、ネットワークインタフェース、および補助ハードドライブがある。これらのリソースは、コアを通じてゲストマシンどうしの間で共有されることも、特定のマシンに専用とすることも可能である。これらのコンポーネントの正確な配置は、セキュリティモデルによって決定することも可能である。例えば、仮想マシンマネージャがディスクからロードされるようにすることにより、ドライバおよびポリシーのアップグレード（おそらくは、新たなハードウェアコンフィグレーションに応じての）を容易にすることが可能である。しかし、自動ハードウェア再設定がセキュリティの脅威となる状況では、システム全体をブートＲＯＭからロードすることが可能である。
【００３９】他のセキュリティ特有のハードウェアコンポーネントをアーキテクチャに組み込むことも可能である。一例としては、２鍵暗号化方式の隠し鍵部分を実装するフュージブル(fusible)ＲＯＭ（すなわち、いったん書き込んだデータを読み出すことができないＲＯＭ）がある。このようなＰＲＯＭがあれば、隠し鍵は、マシン上に可読形式で格納されない。このようなＰＲＯＭは、チャレンジ・レスポンス認証メカニズムを実装するために使用することも可能である。
【００４０】リムーバブルメディアから安全なシステムをブートすることができることは、特に、感染したメディアがサイト付近に運ばれてきたことが知られているときには、許容できないセキュリティの抜け穴となる。既知の良好なメディアからマシンをブートすることができる場合でも、ユーザには、保護されているはずのマシンのセキュリティオプションにアクセスする機会が与えられる。しかし、このオプションを除去すれば、新たなマシンをセットアップすることは困難または不可能になってしまう。
【００４１】この問題点にはいくつかの解決法がある。第１の解決法は、ハードドライブを既存のシステムから設定することができる特殊な「セットアップ」マシンを作成することである。第２の解決法は、リムーバブルメディアからのブートを可能にするハードウェアジャンパを設けることである。第３の解決法は、リムーバブルメディアからブートできないようにする不揮発性ＲＡＭの「ディスエーブル」ビットをを設け、マシンが設定されたらそれをセットすることである。第４の解決法は、マシンが稼働し始めたら置換される特殊なブートＲＯＭをダウンロードまたは挿入することである。どのようにしてリムーバブルメディアからブートできないようにするかに関して、これらの選択肢のうちのいずれを選択するかは、個々のサイトのセキュリティ環境および運用手続きに依存する。コアがロードされ、仮想マシンマネージャが起動した後は、個々の仮想マシンが作成され、そのマシンの仮想の「リムーバブルメディアからブート」オプションを有効にする（これは後でコアから無効にすることが可能）ことによって、それらのオペレーティングシステムは従来のメカニズムによりインストールされる。
【００４２】インストールに手作業の介入を必要とすることは、デスクトップ型の企業環境ではポリシーの本質的な部分かもしれないが、国際的なインフラストラクチャ内に深く埋め込まれたアクティブなネットワークコンポーネントには不適当なことがある。このようなシステムでは、システムソフトウェアのアクティブアップグレードが重要となることがある。このような状況では、ネットワークを通じてコアシステムをロードすることが好ましい場合がある。ブートＲＯＭに復号鍵を埋め込むことにより、権限のあるサーバのみがこの初期イメージを供給することができることを保証することが可能となる。
【００４３】固定システムのインストール手続きにおけるその後のステップは、システムが実際にどのように管理される場合でも本質的に同じままである。ブート可能ハードドライブをフォーマットしパーティションを切るためのユーティリティ（そのマシン上で、または、別のシステム上で実行される）が必要である。ターゲットハードウェア上のブートＲＯＭは、適当なブート環境で再プログラムされなければならない（なお、ゲストシステムが使用するためにオリジナルのＲＯＭをイメージとして保存することも可能である）。ブート可能コアと、仮想マシンマネージャのイメージは、安全なパーティションにインストールしなければならない。
【００４４】不揮発性ＲＡＭデータは、適当なハードウェアおよびセキュリティオプションで設定されなければならない。マシンのセキュリティ鍵を生成し、システムにインストールしなければならない。おそらくは、新たにロードされるパーティション全体が暗号化されることになる。
【００４５】この時点で、システムは、仮想マシンマネージャのマシンにブート可能となる。このマシンからターゲットマシンに、コンフィグレーションを引き継ぐこと、あるいは、ディスクおよびプログラミングの情報を転送することが可能である。
【００４６】個々のゲスト仮想マシンのプロファイルを作成しなければならない。これには次のものが含まれる。
ａ）そのマシンのブートＲＯＭとしてインストールされるイメージ。これは、オリジナルマシンのＲＯＭであるか、あるいは、例えば別のシステムがエミュレートまたは開発される場合には別のソースからのＲＯＭであることが可能である。

ｂ）不揮発性ＲＡＭ要求に対して供給されるデータを含むファイル。これは、最初は空であるか、または、その仮想システムにとって適当な最小限のコンフィグレーションデータを含むことが可能である。
ｃ）リング０またはリング１タスク領域にロードされるアプリケーション。
ｄ）仮想マシンのディスクドライブにマップされるディスクの領域、および、その領域内の保護ブロックの指定（例えば、ブートセクタへの書き込み不可）。
ｅ）専用ハードウェアや外部割り込みソースの指定。
ｆ）専用入出力空間アドレスへのアクセス権。
【００４７】これらのデータが入力された後、ゲスト仮想マシンは、そのブートＲＯＭイメージをマップし、仮想リング０内でリセットベクタから開始することによって起動することができる。最初のブート時に、マシンは新たな未設定のシステムのように見える。ブートＲＯＭセットアッププログラム、または、ブート可能リムーバブルメディアを用いて、不揮発性ＲＡＭシャドウイメージ内のパラメータを設定することができる。例示的なマシンイメージを図１３に示す。
【００４８】その後、個々のマシンは、自分自身のディスク領域をフォーマットし、オペレーティングシステムをインストールする。これは、コンフィグレーションシステムを、ファイルシステムフォーマット（例えば、ＤＯＳやＬｉｎｕｘ）とは独立にする。例示的なディスクレイアウトを図５に示す。
【００４９】補助メモリディスク６０は、パーティションテーブル１０２、実環境パーティションおよびファイルシステム１０４、ファイアウォールパーティション１０６、および、それぞれの仮想環境専用のパーティション１０８ａ～１０８ｄを有する。実環境パーティション１０４は、仮想マシンコアイメージ１０４ａ、仮想マシンマネージャリング０タスクイメージ１０４ｂ、他の実環境特権タスクのイメージ１０４ｃ、仮想マシンマネージャ２０のデータ領域１０４ｄ、および、データ転送マネージャ３０（後述）のスプール領域１０４ｅを有する。仮想マシンマネージャデータ領域１０４ｄは、それぞれの仮想環境ごとに、仕様１４０、ブートＲＯＭイメージ１５０、および不揮発性ＲＡＭエミュレーションイメージ１６０を有する。
【００５０】通常の動作条件下では、電源投入時に、ハードウェアはブートＲＯＭからブートされる。このＲＯＭは、不揮発性ＲＡＭ格納データまたは他のソースからの低レベルマシンパラメータを設定し、（非リムーバブル）ブートディスクにアクセスして仮想マシンコアをロードする。セキュリティモデルに依存して、ＲＯＭは、ロードされるシステムに対する基本的な検証を実行するために、組み込みチェックサムおよび復号機能を有することも可能である。
【００５１】仮想マシンコア初期化プロシージャは、仮想マシンマネージャをリング０タスクとしてロードし、このマネージャを実行キューに入れる。次に、コアは、仮想マシンスケジューラを起動する。これは、メインエントリポイントから仮想マシンマネージャのマシンに入る。
【００５２】この時点で、仮想マシンコアは、タスクごとではなく仮想マシンごとに、オリジナルのコアと同様の「ファンクション」のセットを実行している。ただし、仮想マシンの作成および削除という追加機能がある。これらのファンクションは、リング０コードからのみ直接にアクセス可能であり、次のものを含む。
ａ）物理メモリの割当ておよび解放。
ｂ）割込みおよびトラップハンドラのインストール。
ｃ）仮想マシンの作成および削除。
ｄ）所定のメッセージングプロトコルを用いた仮想マシン環境どうしの間のタスク間通信（リング０およびリング１のみ）
ｅ）割込み無効化サービスによるクリティカルセクションの実装。
ｆ）仮想マシン間のスケジューリングおよびコンテクスト切り替え。
ｇ）登録されたハンドラへの第１レベル割込みディスパッチ。
【００５３】通信プリミティブにより、データは、仮想マシンどうしの間で渡されるが、これは特権プログラムを通じてのみ行われる。仮想マシンマネージャは、外部イベント、入出力空間アドレスへの非リング０アクセス、および、特権命令の実行に対するトラップおよび割込みハンドラをインストールする。また、仮想マシンマネージャは、物理ディスク、キーボード、マウスおよびグラフィクスデバイスにアクセスするためのドライバ、ならびに、不揮発性ＲＡＭおよびリアルタイムクロック情報を読み出すためのドライバも含む。
【００５４】「実」環境内（仮想マシン環境の外部）では、アプリケーションは、システムのコンフィグレーション情報を含む実際のファイルシステムにアクセスすることができる。この場合も、セキュリティモデルに依存して、このブートストラップ環境で実行されるアプリケーションは、仮想マシンのディスク領域にアクセスすることも可能である。これは、例えば、既知の保存されたコピーと対照して仮想セクタ０の内容を確認するため、あるいは、既知のブートブロックウィルスがないかどうかを検査するためである。
【００５５】その後、仮想マシンマネージャは、コアファイルシステム内のコンフィグレーション情報に基づいて、他のゲスト仮想マシンを作成する。これらのマシンは、中央スケジューラの実行リストに追加される。
【００５６】個々のマシンのためのページテーブルの作成および管理、ならびにマシンの制御レジスタのシャドウテーブルの管理を担当するのは仮想マシンマネージャ

である。このように、中央コアは、個々のセキュリティポリシーや管理ストラテジを知っている必要がない。

【００５７】個々のマシンは、それらの機能に適当な状態で起動する。例えば、ほとんどの保護モードマシンは、それらのリセット（仮想）アドレスから起動するが、リング２で実行される。安全なマシン（例えば、ファイアウォールマシン）は、リング０アプリケーションエントリポイントから起動することが必要とされる可能性がある。

【００５８】特権のあるオペレーションやデータへのすべての企図されたアクセスは、仮想マシンマネージャにトラップされる。仮想マシンマネージャは、そのマシンに対して設定されたポリシーと、仮想マシンのシャドウ制御レジスタの現在の状態とに従って結果をエミュレートする。

【００５９】他の特権マシンに引き継がれない外部割込みは、コアによって、仮想マシンマネージャへとスケジュールされ、仮想マシンマネージャは、それらの割込みをゲスト仮想マシンに、それらのマシンの状態に従って分配する。共有リソース（例えばキーボード）について、仮想マシンマネージャは、デバイスの現在の「所有権」(ownership)を追跡し、正しい入力が正しいデバイスに送られることを保証する。ディスクアクセスについては、データは、そのディスク領域にアクセス可能なマシンに送られなければならない（あるいは、そのようなマシンから取得されなければならない）。また、見かけ上の物理アドレスは、マシンごとにマップされなければならない。同様に、不揮発性ＲＡＭのような「エミュレーション」デバイスは、要求元のマシンに適当なデータを返す。

【００６０】一般に、すべてのアプリケーションは、仮想環境で透過的に動作する。各システムは、そのシステム専用のディスクエリアのみを有するように見える。データの暗号化および復号は、透過的に処理される。マシンにマップされるデバイスは、通常のインタフェース（例えば、ＢＩＯＳコンフィグレーションメニュー）を通じてアクセスされるように見える。

【００６１】仮想マシンを動作させる主な動機は、各環境が固有の保護およびセキュリティ領域を有することである。例示的な構成は、次の４つの仮想マシンを含む。

ａ）外部ネットワークにアクセス可能で、ウェブブラウザや電子メールアプリケーションのようなソフトウェアを実行する、安全でない「オープン」なマシン。このようなマシンは、データ窃取アプリケーションが盗んだデータを送信しないように、発信電子メールを阻止することがある。

ｂ）発信電子メールアクセスが可能な低機密性ファイルシステム。これは、ワードプロセッサアプリケーションやその他のウィルスターゲットも実行可能である。

ｃ）機密（または個人に損害を与える）情報を格納し操作するために使用される、外部アクセス不可能な、高機密性マシン。

ｄ）外界に対するファイアウォールとして作用し、イーサネットインタフェースに対する直接制御を有する仮想マシン。このようなマシンは、ＳＭＴＰ(Simple Mail Transfer Protocol)トランザクションをエミュレートし、そのローカルスプールの限り、他のマシンが電子メールを「送信」することを可能にする。ただし、データがマシンを出る前に（例えば、物理マシンから外への、または、仮想マシンどうしの間で仮想ネットワークアダプタを通じて、送信される前に）、明示的なユーザ確認を必要とする。

【００６２】このように、アクセスおよび損害は、個々のプログラムが実行される環境に制限される。この構成でセキュリティを維持することは依然としてある程度のユーザ規律を必要とすることは確かであるが、手続きおよびポリシーが定められるだけではなく守られるという環境では、隔離規則は有効である。実際、ユーザが「作業」(play)環境を作成することを可能にすることによって、ユーザに自分のデスクトップ上での自由度をより多く与えながら、データを不要にさらさないことにより全体のセキュリティを向上させることができる。

【００６３】このモデルはビジネス用途に制限されない。例えば、家庭では、１つの仮想マシンは「子供向け」ウェブブラウザのみを実行し、監視なしのバージョンは、異なるパスワード保護された仮想領域でのみアクセス可能である（従って、暗に、フルバージョンを用いてダウンロードされたデータは他人の眼からは隠されるということになる）。

【００６４】マルチマシン仮想環境モデルを考えると、データを仮想マシンどうしの間で受け渡ししなければならない場合がある。例えば、安全でない環境で受信された電子メールメッセージを、機密領域に渡す必要がある。このようなデータ転送は、リング１アプリケーションを通じて処理される。

【００６５】リング１アプリケーションは、ＮＦＳ(Network File System)によって使用されるのと同様の「シャドウドライブ」を作成し、セキュリティの高い環境のアプリケーションから開始されたオペレーションが、セキュリティの低い環境にアクセスすると、リング１にトラップされるようにする。この方式の１つの利点は、このようなインタフェースを通じてウィルスチェックを適用することができることである。リング１で特別のアプリケーションを動作させることにより、転送コードは、マシン間でデータを探索しマップするためにリング０ゲートコールのセットを使用する。

【００６６】このアプローチの欠点は、共有デバイスが、相異なるマシンからの要求が正しく処理されるとともに適当な正当性チェックとともに処理されることを保証するドライバを通じて、制御されなければならないこ

とである。入出力アクセスは、例えばディスクアドレスを変換し、あるいは、データを読み書きするためのバッファを設けるように、必要に応じて修正されなければならない。

【００６７】本発明は、安全な小型のオペレーティングシステム上で動作することを意図しているため、ドライバはこれに応じてポートされなければならない。さまざまなハードウェアコンポーネントのための最新のドライバを維持しようとする労力は、特に、Ｌｉｎｕｘグラフィクスカードドライバに対するソースレベルサポートがほとんど全くないことにも見られるように、さまざまなメーカによる非開示に対する苦労が増大していることを考えると、過小評価されるべきでない。ＸＦｒｅｅ８６の場合のように、多くの場合、メーカは、ドライバのバイナリバージョンのみを供給している。

【００６８】必要とされるセキュリティが絶対的か相対的かに依存して、実行時間を改善するためのトレードオフを考慮することが可能である。例えば、「カレント」仮想マシン環境に対して、ビデオバッファに用いられるメモリマップドアパーチャへの直接アクセスを可能にすることにより、モニタへの直接書き込みが可能となる。しかし、同じビデオカードの制御レジスタへのアクセスを可能にすると、（潜在的に）アプリケーションがシステムをロックアップさせることが可能になるおそれがある。

【００６９】また、本発明は、コンピューティングデバイスがオープンアクセスネットワーキングフレームワークに埋め込まれている場合に、より安全な相互作用を可能にする。このようなアプリケーションの１つは、アクティブネットワーク(ActiveNetwork)の領域にある。この場合、データストリームがコンピュータネットワークを流れるのとともに処理がデータストリームに適用される。アクティブネットワークの重要な特徴は、処理が事前に固定されておらず、処理されるデータストリームの内容に従って処理ユニットに動的にロードされるデータおよびメソッドを含むことが可能であることである。これらのプログラム自体、データと同じオープンネットワークを通じてプロセッサに転送され、従って、悪意のある第三者による観測や干渉にさらされる。

【００７０】通常のシステムでは、アプリケーション（例えば、Ｊａｖａ対応ウェブブラウザ）は、公衆の安全でないインターネットを通じて、信頼されるソースからモジュールをダウンロードする。セキュリティ（安全性）を高めるには、公開鍵暗号化を使用する。アプリケーションは、モジュールリポジトリへの安全なチャネルを開設しようとする場合、秘密鍵を用いて暗号化されたメッセージをリポジトリに送り、リポジトリは、公開鍵でこのメッセージを復号し、それに応答して、（公開鍵で暗号化した）適当なモジュールを送信する。モジュールは、アプリケーションに到達すると、秘密鍵で復号される。

【００７１】通常、秘密鍵は１つのマシンによって保有され、その上で実行されるアプリケーションどうしで共有される。秘密鍵は、その重要性が高いため、ブートＲＯＭまたは不揮発性ＲＡＭに記憶され、あるいは、それらに記憶された情報から導出されることが多い。この従来の方法の問題点は、悪意のあるアプリケーションもまた鍵を要求することが可能であることにより、他のアプリケーションの活動に対するデータスヌーピング(data snooping)が可能となってしまう。同様に危険なこととして、悪意のあるアプリケーションは、鍵を盗み、それを他の場所にブロードキャストすることができてしまう。

【００７２】安全な仮想環境の作成により、ある仮想環境で実行されるアプリケーションが、別の仮想環境で実行されるアプリケーションに対するスヌーピングをすることができないようになるが、鍵の公然の窃取は妨げられない。

【００７３】鍵の窃取を防ぐため、仮想環境で動作しているアプリケーションにより鍵に対する要求（例えば、所定のアドレスへのコール）がなされると、その要求はリング０コアにトラップされる。その場合、マシンの実際の鍵を提供する代わりに、仮想マシンマネージャ（またはその他の実環境タスク）は、物理マシンの鍵を用いて暗号化されたメッセージを鍵サーバに送り、１回限りのまたは短寿命の秘密鍵を要求する。

【００７４】その後、この短寿命秘密鍵は、セッション中に使用するために、要求元のアプリケーションに渡される。これらの公開鍵および秘密鍵は、個々の仮想環境に固有である。従って、その仮想環境内のすべては、他のすべての仮想環境とは独立の鍵を有し、この鍵は、仮想環境内のアプリケーションが全く知らない間に変更可能である。さらに、コアが実マシンの秘密鍵を隠しているため、この鍵は仮想環境内からは決して見えず、従って、安全である。

【００７５】

【発明の実施の形態】本発明による自動データ装置は、仮想マシン環境の周りにセキュリティシェルを作成すると同時に、仮想環境内からコードを直接に実行する。

【００７６】図２に示すように、装置は、実特権リング０～３の階層を使用してアクセス特権を制御する。これらの実特権リングは、装置の「実」すなわち通常の処理環境からなる。図３のＡは、仮想マシン環境を含む、本発明の基本コンポーネントを示す図であり、図３のＢは、Ａの仮想マシン環境内の仮想リングの階層を示す図である。図３のＡに示すように、実環境内には、タスク切り替え、リソース管理および割込みディスパッチを含むファンクションのセットを実装した小型のコアシステム５０がある。このようなタスクの１つは、最上位特権の実リング内にある仮想マシンマネージャ２０（以下

「ＶＭマネージャ」という）であり、実リングのプロパティを変更するすべての処理はこれに制約される。実環境内には、ＶＭマネージャ２０によって作成された複数の仮想マシン環境４０があり、これらは実特権リング２および３に存在する。

【００７７】仮想環境４０内から見ると、あらゆるものは「実」の幻影を有する。しかし、物理リソースへのコールは実際にはＶＭマネージャ２０によって処理される。ＶＭマネージャ２０は、このようなコールを直接にトラップするか、または、コールをマスクして、アクセスが制限されるようにする。図４は仮想環境専用のメモリのパーティションを示す図である。例えば、仮想環境４０はメモリ４ｂ内のメモリ部分４０′内にある。通常、仮想環境４０内で実行されるプロセスは、アドレス０から始まる連続したアドレスの範囲が見えることを期待する。しかし、仮想環境に割り当てられるメモリの部分は０から始まる可能性は低く、必ずしも連続しているわけでもない。これを調整するため、ＶＭマネージャ２０は、プロセッサのメモリ管理ユニット（ＭＭＵ）のファシリティを用いて、メモリ部分４０′のアドレスをマップし、仮想環境４０内からすべてが通常のように見えるようにする。このようなマッピングは一般に、ページメモリ管理ユニットまたはコントローラによって実行されるため、ＶＭマネージャ２０は、利用可能なときにこのようなツールを利用することが好ましい。

【００７８】同様に、ハードドライブのような補助メモリの場合、仮想環境４０内から絶対ブロックアドレスにアクセスしようとする試みは、ＶＭマネージャ２０によってトラップされ変換されなければならない。仮想環境４０内からは、たとえ複数の「仮想」ドライブが物理的には同じデバイスであっても、ドライブは専用ドライブのように見える。これもまた、透過的なドライブパーティショニングによって実現することができるが、これはIntelアーキテクチャのＰＣによって直接にはサポートされていない。しかし、透過的パーティショニングがハードウェア・ファームウェアによってサポートされている場合、ＶＭマネージャ２０は、「仮想ドライブ」を管理するためにこのようなツールを利用することが好ましい。

【００７９】仮想環境との間のデータ転送は、特権リング１にあるデータ転送マネージャ３０を通じて行われる。仮想環境４０内で実行されるプロセスから見ると、データ転送マネージャ３０は、イーサネットネットワークコネクションのような通信ポートであるように見える。複数の仮想環境がある場合、環境間の通信はデータ転送マネージャを通じて行われ、仮想環境どうしは通信媒体（例えば物理ローカルエリアネットワーク）を通じて接続された別個のエンティティとして互いを認識する。仮想環境４０が物理コンピュータの外部のマシンと通信しようとするとき、情報は、データ転送マネージャ３０からポートあるいはゲートウェイ２２を通じて外界に渡る。このようなゲートウェイの例としては、イーサネット接続を行うイーサネットドライバや、モデムドライバがある。

【００８０】仮想環境４０は、仮想特権リング（図３のＢ）に分かれる。通常、実リングと同数の仮想特権リングがあるが、仮想環境は実リングの残りの部分（すなわち、実リング２および３）のみを含むため、ＶＭマネージャ２０がディスクリプタテーブルの制御権を有する。

【００８１】第１実施例は、さらに２種類のディスクリプタテーブルを有する。第１のものは、仮想環境ディスクリプタテーブル（ＶＥＤＴ：virtual environment descriptor table）である。ＶＥＤＴは、仮想マシン内で実行されるオペレーティングシステムソフトウェアによって作成され、そのシステムによってセグメントに割り当てられた仮想リング番号を有する。仮想マシンに関する限り、ＶＥＤＴは、ＧＤＴまたはＬＤＴとして使用することが可能である。ＶＥＤＴは、仮想環境内の情報を、実リングにではなく仮想リングにマップし、セグメントを、実アドレスにではなく、仮想環境を含むメモリ部分４０′のマップされたアドレスにマップする。

【００８２】追加の２番目の種類のディスクリプタテーブルは、シャドウディスクリプタテーブル（ＳＤＴ：shadow descriptor table）である。ＳＤＴは、ＶＭマネージャによって生成され、ローカル環境内のセグメントを実特権リングの階層にマップする。プロセッサが、仮想環境内から発した命令を実行しているとき、ＳＤＴは、特権を決定するために使用される。実際には、仮想環境内で使用されるＶＥＤＴごとに固有のＳＤＴが生成される。ＳＤＴは、命令がどの特権で実行されるかに基づいて選択される。

【００８３】図６は、第１実施例の実環境および仮想環境の両方に対する特権リングおよびコールゲートを示す図である。第１実施例では、図６に示すように、仮想環境を含む実リング以外の残りの部分は、ＶＥＤＴにおける仮想リング０に関連づけられたすべてのセグメントがＳＤＴにおける実リング２に関連づけられるように分けられる。仮想リング１～３における仮想環境内で実行される命令は、ＳＤＴにより実リング３に割り当てられる。このリング割当ての分配が好ましいのは、仮想環境４０内では、レガシーＯＳが（ほとんどの場合）、例えば入出力オペレーションを処理するためのドライバと同様に、仮想リング０にあるからである。コールゲートが最も多く使用される場合の１つは、このような特権にアクセスする場合である。後述するように、低い特権の圧縮仮想リングから高い特権の圧縮仮想リングへのアクセスを要求するために仮想環境コールゲート１１８（図６）が使用されるときにはＶＭマネージャ２０が介在しなければならないため、非圧縮リングとして仮想リング０を構成することによりＶＭマネージャの介入の必要が

少なくなることにより、処理が高速になる。

【００８４】仮想環境内の仮想特権リングどうしの間の特権規則は、図２に関して説明したような実環境のものと同じである。仮想環境内の命令は、同位または下位の特権の仮想リングに関連づけられた情報およびリソースにアクセスすることができる。上位の特権リングにアクセスしようとする命令は、仮想コールゲート１１８のうちの１つを使用しなければならない。これらの仮想コールゲート１１８はＶＥＤＴ内に現れ、対応するコールゲートがＳＤＴ内にマップされる。

【００８５】第１実施例では、仮想環境内のプロセスから情報を求める要求は、上位、下位、または同位のいずれの特権であるかにかかわらず、ＶＥＤＴではなくＳＤＴを用いて決定される。

【００８６】図７に、ＶＥＤＴ７０および４個のＳＤＴ８０ａ～ｄを示す。（説明のため、この図のディスクリプタテーブルには特権情報のみを示すが、実際には、これらのテーブルは、メモリアドレスのような、ディスクリプタテーブルに通常見られる他の情報も含む。）ＳＤＴ８０ａ～ｄのそれぞれは、仮想特権リングの階層の１つの仮想特権リングに関連づけられる。

【００８７】例えば、仮想リング０で動作する命令が情報を要求するとき、仮想リング０（８０ａ）のＳＤＴが、ディスクリプタ情報のためにプロセッサによって使用される。プロセッサは、この命令を、実リング２で動作する命令と見なす。命令が最上位の仮想特権リングで動作しているため、これは、仮想環境４０全体の情報にアクセス可能である。従って、ＶＭマネージャ２０は、仮想環境内のすべてのセグメントについて、実リング２をＳＤＴ０（８０ａ）に入れる。仮想リング０で動作する命令が、仮想環境内のあるセグメントへのアクセスを要求すると、プロセッサは、命令が動作している実特権リング（実リング２）を、ＳＤＴ０（８０ａ）に記録されている実特権リングと比較する。これらは等しいため、プロセッサはアクセスを許可する。

【００８８】これに対して、仮想リング３で動作する命令は、仮想リング３の外側にアクセスするには、仮想コールゲート１１８を使用しなければならない。仮想リング３で動作する命令に対して、プロセッサは、ＳＤＴ３（８０ｄ）を用いて、特権を決定する。この場合、ＶＥＤＴ７０に示されるように、仮想リング３に関連づけられたすべてのセグメントに対して、ＶＭマネージャ２０は、実リング３を対応するＳＤＴ３（８０ｄ）ロケーションに入れる。こうして、仮想特権リング３内の要求は合法となり、プロセッサはアクセスを許可する。

【００８９】ＳＤＴ３（８０ｄ）内で、ＶＭマネージャ２０は、すべてのセグメントに対して実リング２を仮想リング０に入れる。仮想リング３内の命令が仮想リング０へのアクセスを要求すると、プロセッサは、命令の実リング（実リング３）を、要求される特権（実リング２）と比較し、例外を発行する。この遷移は、コールゲートを通じて実行されるのでなければ不法であるからである。遷移が適当なコールゲートを通じて試みられた場合、遷移は許可される。

【００９０】ある圧縮仮想リングから上位の特権の圧縮仮想リングへのアクセスが要求されると、ＶＭマネージャが介入する。ＳＤＴ３（８０ｄ）内で、ＶＭマネージャ２０は、すべてのセグメントに対して、実リング０を、上位の特権の圧縮仮想リング（リング１および２）に入れる。前と同様に、仮想リング３から仮想リング１または２への遷移がコールゲートを使用せずに試みられた場合、例外が発生する。しかし、実リング０への仮想環境コールゲート要求の実行は、ＶＭマネージャ２０にトラップされる。これは、仮想環境４０内から特権オペレーションまたは特権データにアクセスしようとする場合と同様である。これらの場合、コールのターゲットアドレスが存在するメモリセグメントに要求される実特権レベルは、仮想ゲートの特権レベルにとって十分ではない。これにより、ＶＭマネージャ２０へのトラップが引き起こされるために、ＶＭマネージャ２０は、仮想リングを変更し、新たな仮想環境を反映するように実ＳＤＴテーブルを切り替えることができる。その後、ＶＭマネージャ２０は、その仮想マシンのために設定されたポリシーに従い、要求されたセグメントの対応するＶＥＤＴエントリの内容に基づいて、結果をエミュレートする。

【００９１】シャドウディスクリプタテーブル（ＳＤＴ）８０ａ～ｄは、一度に１個ずつ、または、仮想環境４０の形成時に全部、ＶＭマネージャ２０によって作成される。第１実施例では、ＳＤＴ形成ポリシーは、仮想環境４０の既知のまたは観測される挙動に基づいて、ＶＭマネージャ２０によって適応的に選択される。

【００９２】最も単純なポリシーは、個々の仮想特権リングに対して、そのリング内でプロセスが実行を開始するときに１個のＳＤＴを作成するというものである。このＳＤＴは、セグメントがメモリ部分４０′内でロケーションを変更する（または削除もしくは追加される）か、または、その他の何らかの変更がＶＥＤＴに生じたが、実行中のプロセスはアクセス特権を変更しない場合に、修正される。実行中のプロセスが特権リングを変更すると、このＳＤＴは廃棄され、新たなＳＤＴが生成される。このポリシーは、仮想リング変更が頻繁である場合に、オーバーヘッドは増大するが、維持されるＳＤＴが最小限になるという利点を有する。

【００９３】第２のＳＤＴポリシーは、仮想環境４０の形成時に仮想リングの階層全体に対してＳＤＴを作成するというものである。ＶＥＤＴに変更が生じるたびにすべてのＳＤＴが修正される。このポリシーは、維持されるＳＤＴは増大するが、仮想リングどうしの間の遷移は高速になるという利点を有する。このポリシーは、仮想環境内のプロセスが頻繁にリングを変更する状況に最も

よく適している。

【００９４】第３のＳＤＴポリシーは、上記の２つの混成であり、ＳＤＴは必要に応じて一度に１個ずつ作成されるが、いったん作成された後は、維持され再使用されるというものである。

【００９５】ＶＭマネージャ２０が、仮想環境４０内のレガシーオペレーティングシステムを認識した場合、このレガシーＯＳの特性に適したＳＤＴポリシーが実行される。環境の内容が認識されない場合、ＶＭマネージャ２０はデフォルトポリシー（例えば、第１実施例の第１のポリシー）を選択し、効率が示すところに応じてポリシーを切り替える。

【００９６】複数のディスクリプタテーブルが存在するシステムでは、それぞれをシャドウ化しなければならない。これは、グローバルテーブルとローカルテーブルの両方があるときにも当てはまる。本発明の代表的な実装では、グローバルテーブルは、ＶＭマネージャ２０が仮想マシンどうしの間を切り替えるときに変更され、ローカルテーブルは、仮想マシンのオペレーティングシステムがタスクどうしの間を切り替えるときに変更される。

【００９７】図８は、第２実施例の実環境および仮想環境の両方に対する特権リングおよびコールゲートを示す図である。本発明の第２実施例では、仮想環境は、２個の仮想リングのみを有する。一例として、Intelアーキテクチャマシンでは、標準的なＤＯＳの実装は、２個のリング、すなわち、最上位および最下位の特権のリングのみを使用する。ＶＭマネージャ２０がこのようなレガシーオペレーティングシステムを認識すると、仮想環境４０は、「クイックパススルー」(quick-pass-through)を利用して形成される。図８に示すように、２つの仮想リングは、仮想環境４０を含む２つの実リングに直接にマップされる。その結果、すべてのリングに対するＳＤＴを維持するシャドウディスクリプタテーブル（ＳＤＴ）ポリシーを使用する場合、２個のＳＤＴのみを維持すればよい。そのため、第２実施例では、すべてのシャドウディスクリプタテーブルが環境の作成時に形成されるというＳＤＴポリシーが、好ましいデフォルトとなる。

【００９８】本発明の第３実施例では、シャドウディスクリプタテーブル（ＳＤＴ）を使用する代わりに、ＶＭマネージャ２０は、仮想環境ディスクリプタテーブル（ＶＥＤＴ）を修正して、実特権リングの関連づけを、対応する仮想リングの関連づけの代わりに置き換える。仮想環境内から命令が実行されると、プロセッサは、ＶＥＤＴを用いて、アクセス特権を決定する。

【００９９】仮想環境内からのコードがこの置換を認識しないようにするため、ＶＥＤＴを含むメモリは、仮想環境内からは読むことができないものとしてマークされる。仮想環境４０内からＶＥＤＴを読もうとすると、ＶＭマネージャ２０にトラップされる。ＶＭマネージャ２０は、実特権置換を適当な仮想環境情報でマスクする。

【０１００】ＶＥＤＴへの修正は、２つの方法のうちの一方により実行される。第１の方法では、仮想環境４０内からＶＥＤＴに書き込みをしようとすると、ＶＭマネージャ２０にトラップされる。これにより、ＶＭマネージャは、目的とされたエントリを横取りし、置換エントリを入れる。第１の方法は、ゲストＯＳがＶＥＤＴエントリを確認（読み出しを必要とする）しようとしない場合に好ましい。第２の方法は、仮想環境４０内の命令が、ＶＥＤＴに書き込みをすることを許容し、その後、エントリを適当な実特権リング割当てで上書きするものである。ゲストＯＳがエントリへの書き込みの後にそのエントリを確認する場合、ＶＥＤＴを読み出そうとする試みが、ＶＭマネージャに上書きを実行することを指示するトラップとして使用される。動作時の環境の観測に基づいて、ＶＥＤＴ修正あたりのトラップ数が最小となる方法を選択することが好ましい。

【０１０１】本発明の実施例について説明を続けると、仮想環境との間の情報転送は、実リング１で動作するデータ転送マネージャ（ＤＴＭ）によって処理される。データストリームを管理するため、ＤＴＭは、必要に応じて、データを記憶領域にスプールする（図５、ＤＴＭスプール１０４ｅ）。図９は仮想マシン環境との間の通信のための例示的なコンフィグレーションを示す図であり、ここではＤＴＭ３０を用いた例示的な構成を示す。使用される個々の構成は、ユーザ初期設定、ハードウェア能力、およびマシンの要求のようなファクタに基づく。ゲートウェイ２２は、リング０通信ドライバおよびデータコネクションを含む。

【０１０２】図９のＡに示す第１の構成では、仮想マシン４０は、ＤＴＭ３０をローカルハブ３４と見なす。完全なトランザクションのみが仮想環境どうしの間で伝達されることを保証するため、あるいは、トランザクションの内容を確認するため、ＤＴＭ３０は、さらに、複数のデータブロックをバッファリングするためのスプーリングシステムを実装する。ハブ３４は、いかなる通信プロトコルが要求される場合でも、それを用いて実装することができる。ユーザが希望すれば、スプーリングが通常使用されるため、相異なる仮想環境に対して相異なるプロトコルをシミュレートすることも可能であり、その場合、ＤＴＭ３０は不可視的に変換を行う。

【０１０３】図９のＢに示す第２の構成では、ＤＴＭ３０は、各仮想環境ごとに通信チャネル（ｘｆｅｒ）３２をシミュレートする。チャネルの例としては、モデムコネクションや、ＮＦＳのような仮想ファイルシステムサービスがある。チャネルは、リング０ゲートウェイドライバ２２が実際に接続されている実際のネットワークのように見えることも可能であり、また、何か他のもののシミュレーションであることも可能である。例えば、ゲートウェイは、イーサネットＴＣＰ／ＩＰリンクである

ことが可能であるが、何らかの理由で、ユーザは、ＤＴＭ３０がモデムをシミュレートするということがある。

【０１０４】図９のＣに示す第３の構成では、複数のＤＴＭ３０が動作し、相異なるタスクとして管理される。この場合、ＤＴＭ３０は、仮想マシンどうしの間のスプーリングのために共有スプーリング領域を有することが必要である。この構成は、ユーザが、さまざまな環境固有の通信機能を必要とする場合に適当である。この場合、カスタマイズされたリング１マネージャを設けることが好ましいからである。

【０１０５】ＤＴＭ構成（図９）において、セキュリティと速度との間で選択をしなければならない。最も高速の構成は、ＤＴＭ３０による関与なしにゲートウェイ２２から直接に仮想マシン４０にデータを流すものである。これは、サンドボックスを崩壊させる危険がある。一般に、これにより、ゲートウェイ２２と仮想環境４０の間のインタラクションが可能になるからである。もちろん、ＤＴＭ３０によるプロトコルエミュレーションは、動作を遅くする。最も安全であるが最も低速の構成は、すべてのデータをスプールし、ウィルスがないかどうか、および、データ盗取アプリケーションによる送信がないかどうかを（ＤＴＭ３０やその他の実環境アプリケーションによって）検査することである。ハードウェアの通信能力（例えば、複数のデータコネクション）や、ユーザによって個々の仮想環境に与えられている信頼度に基づいて、複数の構成を組み合わせて使用することも可能である。

【０１０６】本発明の実施例について説明を続けると、システムは、自己の仮想環境から動作するファイアウォールを含むようにセットアップすることが可能である。図１０は仮想環境内から動作するファイアウォールを使用する本発明の例示的なコンフィグレーションを示す図であり、３つの構成例が示されている。ファイアウォールソフトウェアは、本発明のアーキテクチャ用にカスタム開発することも可能であり、あるいは、スタンドアローンマシン用の市販のファイアウォールソフトウェアによることも可能である。各ファイアウォールは、２つの側、すなわち、内側と外側を有する。保護される環境はファイアウォールの内側にある。ファイアウォールの外側には、ネットワークおよび信頼されない環境がある。ファイアウォールのそれぞれの側での通信は互いに隔離される。

【０１０７】図１０に示す３つの構成例は、図９に関して説明したデータ通信構成の変形である。図１０のＡにおいて、ファイアウォール４２を含む仮想マシンは、内側および外側のリンクのために個別のＤＴＭ３０を有する。図１０のＢでは、ＤＴＭ３０は、内側および外側のリンクのための個別のチャネル３２をシミュレートする。図１０のＣでは、ＤＴＭ３０は、ファイアウォールの内側においてローカルハブ３４をシミュレートする一方、専用のＤＴＭ３０がゲートウェイ２２に接続される。これらの構成は単なる例示であり、明らかに、これら以外の多くの組合せが存在する。曲げることができない要件は、ファイアウォール４２の内側と外側の隔離である。ファイアウォール４２は単純な必要物であるため、ファイアウォールが存在する環境には、コンパクトでセキュリティの厳格なオペレーティングシステムが好ましい。

【０１０８】ファイアウォール４２を含む仮想環境は、ファイアウォール４２に与えられる信頼度に依存して、ゲートウェイ２２からの直接データストリームが所望される環境の一例である。さらに、同じく信頼度に依存して、セキュリティ制限の重複を避けるために、ファイアウォール４２の内側のＤＴＭ３０には、データ窃取に対するデータストリームスキャニングのようなセキュリティ制限を少なくすることが好ましい場合がある。

【０１０９】本発明の実施例への追加として、ＤＴＭ３０またはファイアウォール４２は、データ転送が許可される前にユーザの確認を要求するように設定される。その間、データはスプール（図５、ＤＴＭスプール１０４ｅまたはファイアウォールスプール１０６ａ）に格納される。この機能は、特に、データ盗取を検出するために有用である。

【０１１０】図１１は、仮想環境内からディスクパーティションに格納されたデータを暗号化する例示的なコンフィグレーションを示す図である。本発明の実施例について説明を続けると、仮想マシン４０と、補助メモリ６０内の仮想マシンパーティション１０８の間でデータは暗号化される。この暗号化は、仮想マシン４０内からは不可視であり、好ましくは、仮想マシンのパーティション内のすべてのデータを含む。この暗号化を、物理マシンに固有の鍵と組み合わせることにより、盗まれたハードドライブ上のすべてのデータは、覗こうとする他人の目からさらに良好に保護される。仮想マシン環境の形成時のパスワード保護は、さらにセキュリティを向上させる。この保護は、レガシーオペレーティングシステムへの変更や、データが暗号化されているという表示を仮想環境内に有することなしに得られる。

【０１１１】暗号化は、暗号化部分９０および復号部分９２を含むＶＭマネージャ２０（図１１のＡ）によって、または、別個の暗号化・復号フィルタ９４を通じて、実行される。フィルタ９４は、すべてのデータが通る層（レイヤ）として、または、特権実環境アプリケーションを支援する協調的な「ヘルパー」として、実装される。フィルタ９４は、特殊化されたソフトウェア（例えば、リング０または１におけるアプリケーションとして動作する（図１１のＢ））、または、特殊化されたハードウェア（例えば、フュージブルＲＯＭ）とすることが可能である。さらに、相異なるパーティションに対して相異なる暗号化方式を使用することが可能である。同

様の方法は、リムーバブル記憶装置にも使用され、最も極端な場合には、主メモリの仮想環境部分４０′を暗号化するために使用される（これは、不揮発性メインメモリを用いたシステムに有用となる）。

【０１１２】同様に、暗号化は、仮想マシンを出入りするデータストリームに対しても行われる。たとえば、図１２は仮想環境と外部ネットワークの間のデータ転送の暗号化および復号を行う例示的なコンフィグレーションを示す図であるが、図１２のＡにおいて、暗号化部分９０および復号部分９２は、データ転送マネージャ３０に組み込まれ、図１２のＢでは、データは暗号化・復号フィルタ９４を通る。

【０１１３】もう１つの実施例として、本発明は、アクティブネットワークのコンポーネント上で動作するアプリケーションを保護するために有用である。このような構成に対するセキュリティ要求は、より高度な相互信頼が仮定される環境に対して採用されるモデルとは異なる。このことは、例えば、さまざまな文献に記載されているように、安全でないインフラストラクチャで動作するときのKerberos鍵配送方式の制限によって示されている。上記の実施例をアクティブネットワーク環境に組み込んだ場合、データセキュリティは改善される。一例として、再プログラム可能なネットワークの一部を図１４に示す。これは、

・他のネットワーキングエレメントから供給されるダウンロード可能モジュールを実行可能で、公衆の、安全でないインターネットを通じて到達可能な、アクティブネットワークコンポーネント（ＡＮＣ：Active Network Component）２００、

・ダウンロード可能モジュールを含むアクティブリポジトリ（ＡＲ：Active Repository）２１０、

・ネットワーク内の「鍵サーバ」であるＫＳｅｒｖ１（２３０ａ）およびＫＳｅｒｖ２（２３０ｂ）、

・正当な権限のあるアクティブコンポーネントによってのみアクセスされるアプリケーション固有データ（ＡＳＤ：application-specific data）２４０（例えば課金データ）、

を有する。ＡＮＣ２００は、暗号化鍵が必要な仮想環境４０を含むマシンからなる。

【０１１４】このような構成では、２つの異なるセキュリティ領域がある。第１のセキュリティ領域に、アクティブネットワークでのＡＮＣマシンの認証である。第２のセキュリティ領域は、ＡＮＣ２００にロードされるアクティブコンポーネントの検証である。ＡＮＣ２００上に、本発明によって提供されるような安全な環境がなければ、第１のセキュリティは達成することができない。さらに、ダウンロードされたコンポーネントが移動中に、またはＡＮＣ２００上で、悪意のある第三者による修正を受けた場合、第２のセキュリティは保証することができない。本発明の安全な構成を用いた場合、周知の２鍵暗号方式を用いて、どのようにして両方のレベルの認証が可能となるかを以下で説明する。

【０１１５】本発明によれば、アクティブエレメントはＮａｍｅ（名前）によって表される。Ｎａｍｅは、マシンまたはアプリケーションを表すことが可能である。ＤＮＳ（Domain Name System、分散型インターネットディレクトリサービス）のように、階層的にドメインとして構造化されたＮａｍｅの空間が考えられるかもしれないが、マシンに対するアクティブなＮａｍｅがそのインターネット構成と関係を有する必要はない（すなわち、Ｎａｍｅは、異なる時点では異なる物理インスタンスに関連づけられる可能性のある抽象的エンティティである）。

【０１１６】それぞれのＮａｍｅには、そのＮａｍｅをインスタンス化しているエンティティにのみ知られるＫｅｙ（秘密鍵）と、外界に知らされるＣｌｅｆ（公開鍵）が関連づけられる。通常のモデルに従って、Ｋｅｙで暗号化されたデータは、Ｃｌｅｆで（のみ）復号可能であり、その逆も成り立つ。

【０１１７】ＡＮＣマシン２００は、設定されると、権限３項組｛Ｎａｍｅ，Ｃｌｅｆ，Ｋｅｙ｝が与えられる。また、少なくとも１つのアクセス２項組｛Ｎａｍｅ，Ｃｌｅｆ｝と、鍵サーバマシン２３０ａ／ｂの対応するＩＰ情報｛マシン名，アドレス｝（これは、ＡＮＣマシン２００が鍵問合せを行うために使用することが可能）も供給される。同時に、ＡＮＣマシン２００のアクセス２項組｛Ｎａｍｅ－ＡＮＣ，Ｃｌｅｆ－ＡＮＣ｝が鍵サーバ２３０ａに入力される。秘密鍵の生成およびインストールに関するすべてのオペレーションは、安全に実行される。

【０１１８】仮想環境４０は、固有のＣｌｅｆ（公開鍵）を、秘密のＫｅｙ（秘密鍵）とともに受信する。このＫｅｙは、ＡＮＣ２００が鍵サーバ２３０ａに対して要求する用途限定の鍵（Ｋｅｙ－ＡＮＣ）である。好ましくは、仮想環境４０内のアプリケーションがマシンの秘密鍵に関連づけられたアドレスをコールするときに最初の要求が行われ、これが自動的にリング０コアにトラップされる。実環境の暗号化を利用して、限定用途の秘密鍵に対してコアからなされる要求は、実際のハードウェアの秘密鍵を用いて符号化される。鍵サーバが応答すると、限定用途秘密鍵Ｋｅｙは仮想環境（もともとコールがなされたアドレスにあるかのように）に渡されることにより、その内部のアプリケーションは、その限定用途Ｋｅｙを利用して、アクティブリポジトリ２１０に対してモジュールを要求する。このような方式で標準となっているように、公開鍵で符号化された情報は、秘密鍵を用いてのみ復号可能であり、秘密鍵で符号化された情報は、公開鍵を用いてのみ復号可能である。

【０１１９】限定用途秘密鍵を取得することは、仮想環境４０内のアプリケーションの知り得る範囲外で行われ

る。鍵が取得されると、その仮想環境４０を出入りするデータストリームは独自に符号化することが可能となるため、他の環境で実行される悪意のあるコードが、符号化されたアプリケーションのセキュリティを危険にさらすことはできなくなる。そのアプリケーションの環境の秘密鍵を危険にさらすことは不可能であるためである。

【０１２０】何らかの制御メカニズム（例えば、ＣＯＲＢＡ(Common Object Request Broker Architecture)）を用いて（ただし、以下のプロトコルを実装して）、アクティブリポジトリ（ＡＲ）２１０は、ＡＮＣ２００でインスタンス化された仮想環境に新たなプログラムをダウンロードするよう命令される。ＡＲ２１０は、ＡＮＣ２００のネットワークロケーションを通知されるか、または、ある位置探索・インスタンス化（安全な）メカニズムによりそれを決定する。何らかの方法で、ＡＲ２１０は、ＡＮＣ２００のネットワークアドレスを知る。また、ＡＲ２１０は、鍵サーバ２３０ａのネットワークアドレスも知る。

【０１２１】以下は、限定用途Ｋｅｙが取得された後、モジュールを安全にダウンロードするのに必要なトランザクションの一例である。

【０１２２】１．ＡＲ２１０は、ＫＳｅｒｖ１（２３０ａ）に次の要求を送る。

＜Ｎａｍｅ－ＡＲ　＜Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝？＞$_{Key-AR}$＞$_{Clef-KServ1}$

ただし、＜ｙ＞$_{x}$は、鍵ｘで暗号化された項目ｙを意味し、？は、この要求に対する応答として返されるべきデータを示す。秘密鍵が真に秘密のままであるという仮定の下では、ＫＳｅｒｖ１（２３０ａ）以外のエンティティはこのメッセージの内容を見ることができない。

【０１２３】２．ＫＳｅｒｖ１（２３０ａ）は、このパケットを受信すると、次のオペレーションを実行する。

＜＜Ｎａｍｅ－ＡＲ　＜Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝？＞$_{Key-AR}$＞$_{Clef-KServ1}$＞$_{Key-KServ1}$

＜＜ｙ＞$_{Key}$＞$_{Clef}$＝ｙであるため、これは次のように簡単化される。

Ｎａｍｅ－ＡＲ　＜Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝？＞$_{Key-AR}$

Ｎａｍｅ－ＡＲの公開鍵を適用することにより次が得られる。

＜＜Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝？＞$_{Key-AR}$＞$_{Clef-AR}$

これは次のように簡単化される。

Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝？

これにより、パケットが、主張されたＮａｍｅエンティティからのものである場合に限り、要求が取得される。

【０１２４】３．次に、鍵サーバ２３０ａは、Ｎａｍｅ－ＡＮＣに対するＣｌｅｆを調べ、自己のＫｅｙと、ＡＲ２１０のＣｌｅｆで暗号化された応答を次のように形成する。

＜Ｎａｍｅ－ＫＳｅｒｖ１　＜Ｃｌｅｆ（Ｎａｍｅ－ＡＮＣ）＝Ｃｌｅｆ－ＡＮＣ＞$_{Key-KServ1}$＞$_{Clef-AR}$

これにより、ＡＲ２１０のみがこの応答を読むことができることが保証される。一見したところ、実際には公開された情報であるものを暗号化することは奇妙に思われるかもしれない。しかし、このようにすることは、公開鍵であっても、権限のあるエンティティにのみ公開されていることを意味し、セキュリティが多少向上する。さらに重要なことは、ＡＲ２１０のクライアントがＡＮＣマシン２００に関心を持っていることを、鍵サーバ２３０ａの「付近」をスヌープしている者が推論することができないようにすることである。このレベルでは、些細なことであっても、競争相手に何らかの推論やビジネス上の有利な立場を与えてしまうことがあるからである。

【０１２５】４．ＡＲ２１０は、明らかな復号を実行し、ＡＮＣマシン２００のアクセスコードを最終的に取得する前に、応答中のＮａｍｅが既知の鍵サーバに対応することをチェックする。

【０１２６】ＡＮＣ２００に送られる新たなプログラムが、そのソフトウェアの供給元からの機密情報への安全なアクセス（例えば、ＡＳＤ２４０へのアクセス）を必要とし、ＡＳＤ２４０のプロバイダが、スヌーピングに対してＡＲ２１０の長期記憶装置を信頼してない場合（また、おそらく、ＡＳＤ２４０のプロバイダは、アプリケーションがロードされることになるＡＮＣのＮａｍｅのセットを事前には知らない）、ＡＲ２１０は、ＡＲ２１０およびＡＳＤ２４０によって相互に信頼されている鍵サーバ（２３０ｂ）に次のような第２の問合せを送り、新たな３項組を要求する。

【０１２７】５．　＜Ｎａｍｅ－ＡＲ　＜Ｎｅｗ（Ｎａｍｅ－ＳｅｃＡｐｐ）＝？＞$_{Key-AR}$＞$_{Clef-KServ2}$

ただし、Ｎａｍｅ－ＳｅｃＡｐｐは、ＡＮＣ２００上でインスタンス化されようとしているアプリケーションのＮａｍｅである。

【０１２８】６．ＫＳｅｒｖ２（２３０ｂ）は、Ｃｌｅｆ－Ａｒ（これは、ＫＳｅｒｖ１（２３０ａ）から取得しなければならないかもしれない）を用いて要求を抽出し、新たな認証３項組を生成する。ＫＳｅｒｖ２（２３０ｂ）は、Ｎａｍｅ部分およびＣｌｅｆ部分を自己のデータベースに格納しており、３項組全体をＡＲに返す。

＜Ｎａｍｅ－ＫＳｅｒｖ２　＜｛Ｎａｍｅ－ＳｅｃＡｐｐ，Ｃｌｅｆ－ＳｅｃＡｐｐ，Ｋｅｙ－ＳｅｃＡｐｐ｝＞$_{Key-KServ2}$＞$_{Clef-AR}$

もちろん、この場合、この応答は暗号化されることが重要である。より強力なモデルでは、これをさらにＣｌｅｆ－ＡＮＣで暗号化することにより、ＡＲ２１０がデータを読み出せないようにすることが可能である。

【０１２９】７．ＡＲは、ＫＳｅｒｖ１（２３０ａ）から返された鍵を用いて、再プログラミング情報をＡＮＣ２００に送る。

＜Ｎａｍｅ－ＡＲ ＜Ｎｅｗ Ｐｒｏｇｒａｍ＞Key-AR＞Clef-ANC

【０１３０】８．このパケットは、ＡＮＣ２００の「ファイアウォール」レベル４２で横取りされ、限定用途のＫｅｙ－ＡＮＣにアクセス可能な権限のある仮想マシン環境４０に渡される。

＜＜Ｎａｍｅ－ＡＲ ＜Ｎｅｗ Ｐｒｏｇｒａｍ＞Key-AR＞Clef-ANC＞Key-ANC

これは、次のように簡単化される。

Ｎａｍｅ－ＡＲ ＜Ｎｅｗ Ｐｒｏｇｒａｍ＞Key-AR

この時点で、送信側エンティティのＮａｍｅが明らかになる（Ｎａｍｅ－ＡＲ）。セキュリティモデルに依存して、このＮａｍｅは、権限のあるリストと比較してチェックされるか、または、オープンシステムでは額面通りにとられる。

【０１３１】９．ステップ１～４と全く同じステップを用いて、ＡＮＣ２００はＣｌｅｆ－ＡＲを取得し、Ｎｅｗ Ｐｒｏｇｒａｍ情報を回復する。

【０１３２】新プログラム（Ｎｅｗ Ｐｒｏｇｒａｍ）がリポジトリに安全にリンクされるように、ＡＲ２１０は、権限３項組情報を（おそらくは、ＫＳｅｒｖ２（２３０ｂ）の名前およびそのロケーション情報とともに）ＡＮＣ２００に送る。ＡＲ２１０は、この情報のコピーを保持しないように信頼される（結局、ＡＲ２１０は、格納されたアプリケーションを修正しないように信頼される）。モデルに依存して、この新たなアプリケーション固有の鍵は、新たにロードされるコードに渡されるか、または、仮想環境の外部に保持されることが可能である。Ｋｅｙ－ＳｅｃＡｐｐおよびＣｌｅｆ－ＳｅｃＡｐｐを用いて、新たなアプリケーションは、ＡＳＤ２４０がＣｌｅｆ－ＡＳＤを知ることを必要とせずに、ＫＳｅｒｖ２（２３０ｂ）を通じてＡＳＤ２４０への通信チャネルを確立することができる。このメカニズムにより、ＫＳｅｒｖ２（２３０ｂ）およびＡＳＤ２４０は、ＡＮＣ２００に置かれた信頼とは独立に許可を取り消すことが可能となる。

【０１３３】なお、このモデルは、仮想環境４０内で動作する個々のアプリケーションが特別な方法で構成されることや、特別の言語で書かれることを必要としないことに注意すべきである。さらに、これらと同じ方法を用いて、仮想環境をサービスする暗号化サブルーチン（図１２）に、固有の鍵セットを提供することも可能である。例えば、ＤＴＭ３０または暗号化・復号フィルタ９４が、仮想環境４０を出入りするデータ転送の暗号化および復号を行うために限定用途の鍵を使用することが可能である。このような場合、仮想環境４０とＤＴＭ３０またはフィルタ９４との間を流れる暗号化されていないデータは、他のすべての仮想環境にはアクセス不可能である。

【０１３４】本発明の技術思想および技術的範囲を離れることなく、本発明のさまざまな変形例を考えることが可能であることはいうまでもない。

【図面の簡単な説明】

【図１】Ａは、通常のメモリに記憶されたセグメントおよびディスクリプタテーブルを示す図であり、Ｂは、Ａのディスクリプタテーブルの内容を示す図である。

【図２】階層的特権保護方式を用いたマシン内の特権リングおよびコールゲートを示す模式図である。

【図３】Ａは、仮想マシン環境を含む、本発明の基本コンポーネントを示す図であり、Ｂは、Ａの仮想マシン環境内の仮想リングの階層を示す図である。

【図４】仮想環境専用のメモリのパーティションを示す図である。

【図５】補助メモリディスクパーティションレイアウトを示す図である。

【図６】第１実施例の実環境および仮想環境の両方に対する特権リングおよびコールゲートを示す図である。

【図７】第１実施例の仮想環境ディスクリプタテーブルおよびシャドウディスクリプタテーブルを示す図である。

【図８】第２実施例の実環境および仮想環境の両方に対する特権リングおよびコールゲートを示す図である。

【図９】仮想マシン環境との間の通信のための例示的なコンフィグレーションを示す図である。

【図１０】仮想環境内から動作するファイアウォールを使用する本発明の例示的なコンフィグレーションを示す図である。

【図１１】仮想環境内からディスクパーティションに格納されたデータを暗号化する例示的なコンフィグレーションを示す図である。

【図１２】仮想環境と外部ネットワークの間のデータ転送の暗号化および復号を行う例示的なコンフィグレーションを示す図である。

【図１３】例示的なマシンイメージを示す図である。

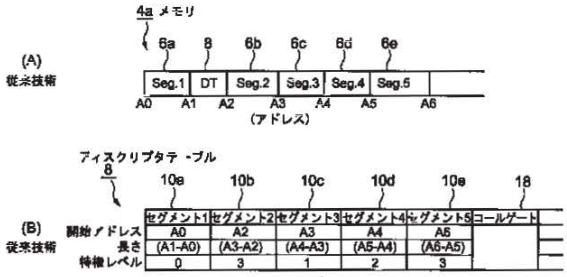
【図１４】アクティブネットワークコンピューティングのための例示的なネットワークレイアウトを示す図である。

【符号の説明】

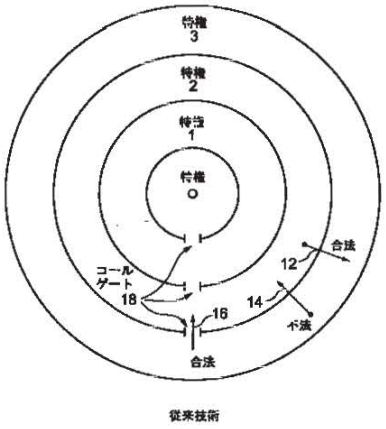
４ａ　メモリ
６ａ～６ｅ　セグメント
８　ディスクリプタテーブル
１０ａ～１０ｅ　ディスクリプタ
２０　仮想マシンマネージャ
２２　ゲートウェイ
３０　データ転送マネージャ
３４　ローカルハブ
３２　通信チャネル
４０　仮想環境
４２　ファイアウォール
５０　コアシステム

６０　補助メモリディスク
７０　ＶＥＤＴ
８０　ＳＤＴ
９０　暗号化部分
９２　復号部分
９４　暗号化・復号フィルタ
１０２　パーティションテーブル
１０４　実環境パーティションおよびファイルシステム
１０４ａ　仮想マシンコアイメージ
１０４ｂ　仮想マシンマネージャのリング０タスクイメージ
１０４ｃ　他の実環境特権タスクのイメージ
１０４ｄ　仮想マシンマネージャのデータ領域
１０４ｅ　データ転送マネージャのスプール領域
１０６　ファイアウォールパーティション
１０６ａ　ファイアウォールスプール
１０８ａ～１０８ｄ　仮想環境専用のパーティション
１１８　仮想環境コールゲート
１４０　仕様
１５０　ブートＲＯＭイメージ
１６０　不揮発性ＲＡＭエミュレーションイメージ
２００　アクティブネットワークコンポーネント（ＡＮＣ）
２１０　アクティブリポジトリ（ＡＲ）
２３０　鍵サーバ
２４０　アプリケーション固有データ（ＡＳＤ）

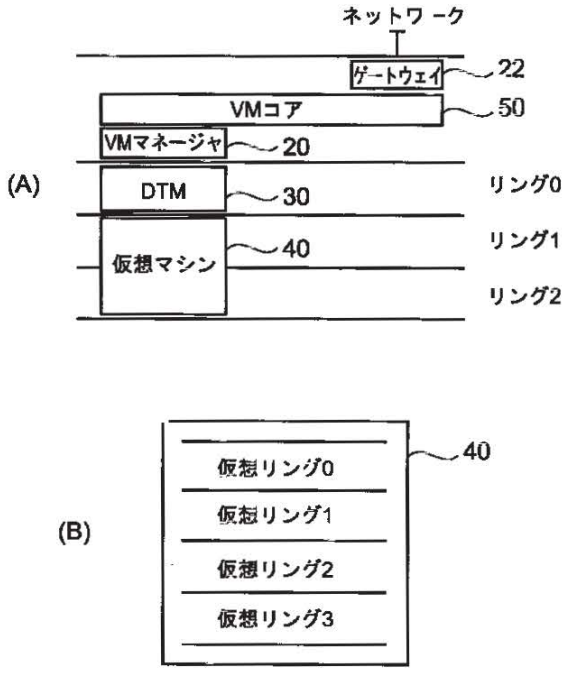
【図１】



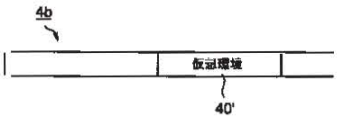
【図２】



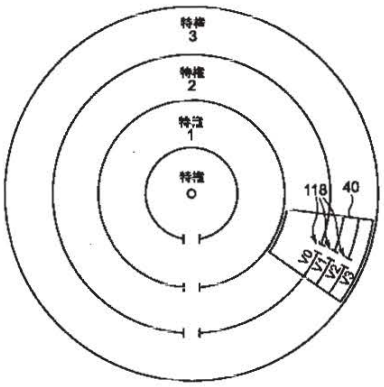
【図３】



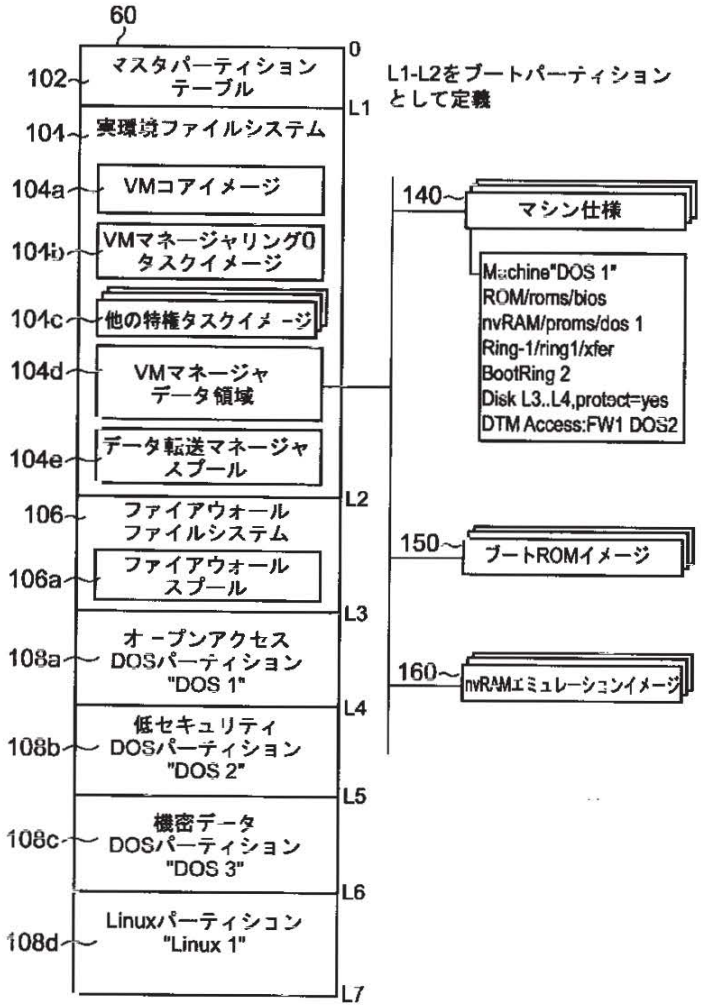
【図４】



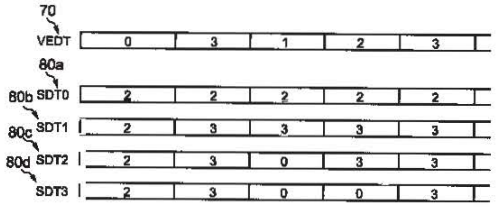
【図６】



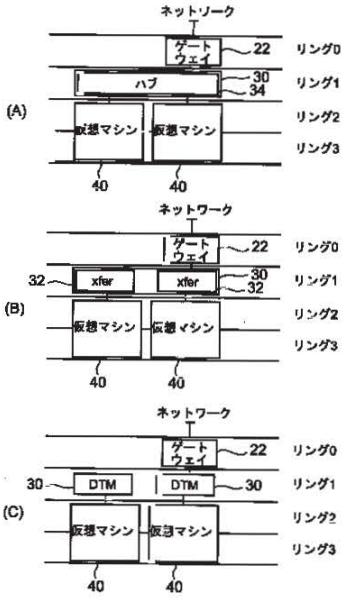
【図5】



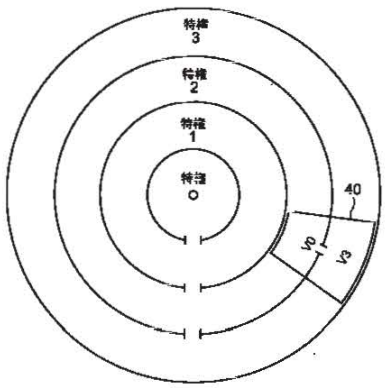
【図7】



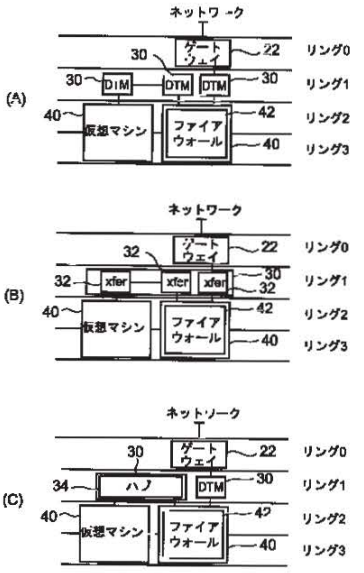
【図9】



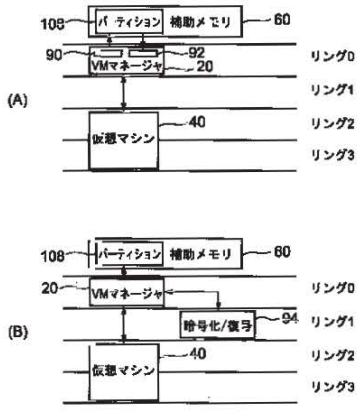
【図8】



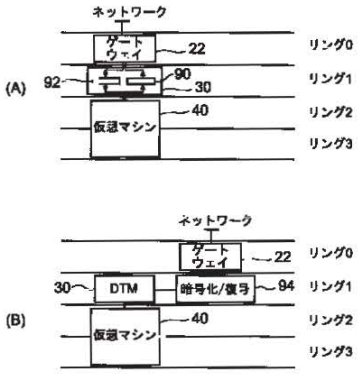
【図10】



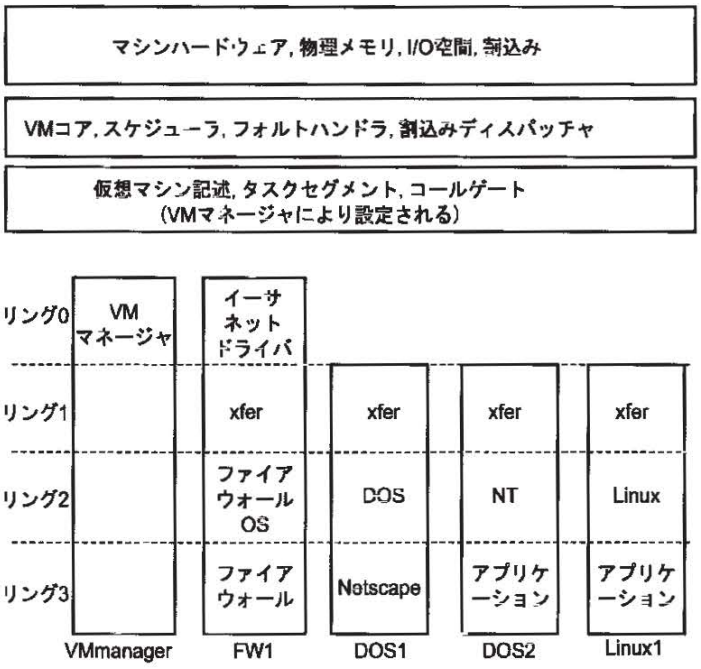
【図11】


108
パーティション
補助メモリ
60
90
92
VMマネージャ
20
リング0
リング1
(A)
仮想マシン
40
リング2
リング3
108
パーティション
補助メモリ
60
20
VMマネージャ
リング0
暗号化/復号
94
リング1
(B)
仮想マシン
40
リング2
リング3

【図12】


ネットワーク
ゲートウェイ
22
リング0
92
90
30
リング1
(A)
仮想マシン
40
リング2
リング3
ネットワーク
ゲートウェイ
22
リング0
30
DTM
暗号化/復号
94
リング1
(B)
仮想マシン
40
リング2
リング3

【図13】


マシンハードウェア, 物理メモリ, I/O空間, 割込み
VMコア, スケジューラ, フォルトハンドラ, 割込みディスパッチャ
仮想マシン記述, タスクセグメント, コールゲート
(VMマネージャにより設定される)
リング0
リング1
リング2
リング3
VM マネージャ
イーサネットドライバ
xfer
xfer
xfer
xfer
ファイアウォール OS
DOS
NT
Linux
ファイアウォール
Netscape
アプリケーション
アプリケーション
VMmanager
FW1
DOS1
DOS2
Linux1

【図14】