(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3546678号
(P3546678)
(45)発行日　平成16年7月28日(2004.7.28)　(24)登録日　平成16年4月23日(2004.4.23)

(51)Int.Cl.[7]　F I
G06F 9/46　G06F 9/46 350
G06F 9/445　G06F 9/06 610K

請求項の数 8　(全 39 頁)

(21)出願番号　特願平10-8299
(22)出願日　平成10年1月20日(1998.1.20)
(65)公開番号　特開平11-149385
(43)公開日　平成11年6月2日(1999.6.2)
審査請求日　平成12年12月20日(2000.12.20)
(31)優先権主張番号　特願平9-248178
(32)優先日　平成9年9月12日(1997.9.12)
(33)優先権主張国　日本国(JP)

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(74)代理人　100075096
弁理士　作田　康夫
(72)発明者　関口　知紀
神奈川県川崎市麻生区王禅寺１０９９番地
株式会社日立製作所　システム開発研究所内
(72)発明者　新井　利明
神奈川県川崎市麻生区王禅寺１０９９番地
株式会社日立製作所　システム開発研究所内

最終頁に続く

(54)【発明の名称】マルチＯＳ構成方法

(57)【特許請求の範囲】
【請求項１】
１台の計算機上で複数のオペレーティングシステム（以下「ＯＳ」）を並行して動作させるためのマルチＯＳ構成方法であって、
第１のＯＳを起動する手段が、前記計算機が有する主記憶領域から第２のＯＳが使用する第２の記憶領域を除いた第１の記憶領域を用いて、前記第２の記憶領域が前記第１のＯＳにより管理されないように前記第１のＯＳを起動し、
前記第２のＯＳをロードする手段が、前記第２の記憶領域を前記第１のＯＳが管理する仮想記憶空間にマッピングし、
前記第１のＯＳがファイル読込み手順を用いて前記第２の記憶領域に前記第２のＯＳを読み出し、
前記ロードする手段が、前記第２の記憶領域の前記第１のＯＳが管理する仮想記憶空間へのマッピングを解除し、
前記第１のＯＳと前記第２のＯＳとを前記計算機上で実行することを特徴とするマルチＯＳ構成方法。
【請求項２】
割り込み処理手段が、
割り込み発生時に割り込み要因に基づいて該割り込みを処理すべきＯＳを決定し、
該割り込み発生時に該計算機で実行されていたＯＳと前記決定したＯＳとが異なる場合に前記決定したＯＳのモジュールを呼び出し、

前記決定したＯＳによる前記割り込みの処理が終了した後に、前記実行されていたＯＳの実行の再開を指示することを特徴とする請求項１記載のマルチＯＳ構成方法。
【請求項３】
割り込み処理手段が、
割り込み発生時に割り込み要因に基づいて該割り込みを処理すべきＯＳを決定し、
前記決定したＯＳのモジュールを呼び出し、
前記決定したＯＳによる前記割り込みの処理が終了した後に、前記割り込み発生時に実行されていたＯＳの実行の再開を指示することを特徴とする請求項１記載のマルチＯＳ構成方法。
【請求項４】
前記実行されていたＯＳの方が前記決定したＯＳよりも優先度が高く設定されている場合、前記割り込み処理手段が前記割り込みの処理を前記実行されていたＯＳによる演算装置の専有が終了するまで待つことを特徴とする請求項２記載のマルチＯＳ構成方法。
【請求項５】
前記第一のＯＳが、該第一のＯＳの回復不能な障害による停止を前記割り込み処理手段に通知し、
前記割り込み処理手段が、
前記第一のＯＳの停止時に前記第二のＯＳに割り当てられたＩ／Ｏ装置の割り込みを許可し、
前記第二のＯＳの割り込み処理終了時に前記第一のＯＳの停止を判定し、
前記第一のＯＳが停止している場合は更に前記第二のＯＳへの割り込みを待つことを特徴とする請求項２、３及び４のうちいずれか一つに記載のマルチＯＳ構成方法。
【請求項６】
前記割り込み処理手段が、前記第一のＯＳ及び前記第二のＯＳの双方から参照可能な記憶領域に格納されたモジュールであることを特徴とする請求項５記載のマルチＯＳ構成方法。
【請求項７】
前記割り込み処理手段によって呼び出されるモジュールは、前記決定されたＯＳでの割り込み処理を行うモジュールであることを特徴とする請求項２及び３記載のうちいずれか一つのマルチＯＳ構成方法。
【請求項８】
一つのプロセッサで前記第１及び前記第二のＯＳを動作させることを特徴とする請求項１記載のマルチＯＳ構成方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は一台の計算機上で複数のオペレーティングシステムを稼働させるマルチＯＳ構成方法に関する。
【０００２】
【従来の技術】
通常の計算機では１つのオペレーティングシステムが動作し、それが計算機のプロセッサ、メモリ、および、二次記憶装置等の計算機資源を管理し、計算機が効率良く動作できるように資源スケジュールを実施している。オペレーティングシステムには様々な種類がある。バッチ処理に優れるものや、ＴＳＳ（ＴｉｍｅＳｈａｒｉｎｇ　Ｓｙｓｔｅｍ）に優れるもの、ＧＵＩ（Ｇｒａｐｈｉｃａｌ　Ｕｓｅｒ　Ｉｎｔｅｒｆａｃｅ）に優れているものなど様々である。
【０００３】
一方で、これら複数あるオペレーティングシステムを１台の計算機で同時に実行したいというニーズがある。例えば、大型計算機においては、実際の業務に伴うオンライン処理を実行するオペレーティングシステムと、開発用のオペレーティングシステムを一台の計算

機で動作させたいという要求がある。あるいは、ＧＵＩの整っているオペレーティングシステムと、実時間性に優れているオペレーティングシステムを同時に稼働させたい等という要求もある。

【０００４】

しかしながら、個々のオペレーティングシステムは、単独で計算機資源の管理を実施することを仮定しており、複数のオペレーティングシステムの共存は、何らかの機構なしには不可能である。

【０００５】

一台の計算機上で複数のオペレーティングシステムを動作させる機構としては、大型計算機で実現されている仮想計算機方式（ＯＳシリーズ第１１巻　ＶＭ、岡崎　世雄他著、共立出版株式会社）がある。仮想計算機方式では、仮想計算機制御プログラムが全ハードウェア資源を占有して管理し、それを仮想化して仮想計算機を構成する。仮想計算機を構成する制御部は、物理メモリ、入出力機器装置、外部割り込み等を仮想化する。

【０００６】

例えば、分割された物理メモリは、各仮想計算機に対してはあたかも０番地から始まる物理メモリのように振舞い、入出力装置を識別する装置番号も同様に仮想化されている。更に、磁気ディスクの記憶領域も分割して磁気ディスク装置の仮想化まで実現している。

【０００７】

各オペレーティングシステムは、制御プログラムにより構築された仮想計算機上で実行されるように制御プログラムによりスケジュールされる。　しかし、大型計算機における仮想計算機方式では、計算機資源を完全に仮想化、および、分割しようとするため、仮想計算機を構成する制御部分が複雑であり問題である。

【０００８】

また、特別なハードウェア支援がない場合、仮想計算機上で動作するオペレーティングシステムが発行する制御レジスタの設定、入出力命令等の特権命令は、仮想計算機制御プログラムによりエミュレートしなければならないため、オーバーヘッドが大きくなり問題である。実際、仮想計算機を実装している大型計算機では、仮想計算機用に特別なプロセッサ機能やマイクロコード等のハードウェアを追加してオーバーヘッドの削減を図っている。仮想計算機方式は、完全に計算機資源を仮想化することを目的としているため複雑であり、更に、仮想計算機の高性能化のためには特殊なハードウェア機構が必要であり問題である。

【０００９】

一方、一台の計算機で複数のオペレーティングシステムのインターフェイスを提供する技術として、マイクロカーネルがある。マイクロカーネルでは、マイクロカーネルの上に、ユーザに見せるオペレーティングシステム機能を提供するオペレーティングシステムサーバを構築し、ユーザはそのサーバを経由して計算機資源を利用する。オペレーティングシステム毎のサーバを用意すれば、ユーザに様々のオペレーティングシステム環境を提供することができる。

【００１０】

しかし、マイクロカーネル方式では、オペレーティングシステムサーバをマイクロカーネルに合わせて新規に構築する必要がある。多くの場合、現在あるオペレーティングシステムをマイクロカーネル上で動作するように変更することになるが、スケジューリング、メモリ管理等のカーネルの中枢部分も変更することになり、変更箇所が多く、また、変更箇所がオペレーティングシステムの中枢部分に及ぶため変更作業が複雑で容易でなく問題である。

【００１１】

また、オペレーティングシステムサーバはマイクロカーネルのサービスを利用することになるが、これは通常のオペレーティングシステムではないことであり、オーバーヘッドとなり性能低下をもたらす。

【００１２】

【発明が解決しようとする課題】
従来の仮想計算機方式は、複数のオペレーティングシステムを同時に動作させるために、全計算機資源を仮想化する手法によっていた。しかし、この方式では制御プログラムが複雑になる問題がある。更に、この方式では特権命令のエミュレーションが必要となるため、性能を得るには特別なハードウェアが必要であり問題である。
【００１３】
本発明は、オペレーティングシステムの初期化処理部分、割り込み管理部分の変更、および、割り込み管理プログラムの追加により、特別なハードウェアなしで複数オペレーティングシステムの同時実行を実現する。本発明では、特権命令のエミュレーションは不要なため、各オペレーティングシステムの実行に新たなオーバーヘッドは伴わない。
【００１４】
本発明によれば、第一のオペレーティングシステムを補完する機能を容易に追加でき、高機能な計算機システムの構築が可能になる。更に、デバイスドライバとは異なり、第一のオペレーティングシステムとは全く独立して動作する機能を組み込むことができるため、第一のオペレーティングシステムに依存しない高信頼化機能を追加することも可能になる。
【００１５】
また、マイクロカーネル方式で複数のマルチオペレーティングシステム環境を構成する方法では、それぞれのオペレーティングシステムのインターフェイスを提供するオペレーティングシステムサーバの構築が難しいという問題がある。本発明によれば、オペレーティングシステムへの変更は初期化部分と割り込み管理部分のみに限定されるため、簡単にマルチオペレーティングシステム環境を構成できる。
【００１６】
【課題を解決するための手段】
本発明は、第一のＯＳにおいて、第二のＯＳが必要とする物理メモリ、外部デバイス等の計算機資源を予約し、どちらのＯＳからも独立した管理プログラムが外部割り込みを横取りして、割り込み要因によりどのＯＳの割り込みハンドラを起動すべきか決定し、ＯＳの実行状態により割り込みハンドラを起動するタイミングを決定して、それの基づいて各ＯＳの割り込みハンドラを起動することにより、２つのＯＳを一台の計算機で動作させる。
【００１７】
【発明の実施の形態】
本発明の実施の形態について説明する。
【００１８】
以下、図面を用いて本発明の実施の形態について説明する。図１は本発明の実施の形態における計算機１００の構成を示す図である。
【００１９】
計算機１００は、プロセッサ１０１、主記憶装置１０２、バス１０９、割り込み信号線１１０、クロック割り込み生成器１１１、割り込み制御装置１１２、ブート手順を格納している記憶装置１１８、および、割り込みバス１１９より構成されている。
【００２０】
割り込み信号線１１０は、外部の入出力機器と割り込み制御装置１１２を接続している。外部機器が割り込みを発生すると、割り込み信号線１１０を経由して割り込み制御装置１１２が信号を受けとり、割り込み制御装置１１２は、この信号を数値化して、割り込みバス１１９を介してプロセッサ１０１に渡す。
【００２１】
クロック割り込み生成器１１１は、周期的な割り込みを生成する。
【００２２】
割り込み制御装置１１２は、外部機器からの割り込み要求を受け付け、要求元にしたがって数値化された割り込み信号を生成し、プロセッサ１０１に送る。また、プロセッサ１０１からの指示により、特定の機器からの割り込み信号をプロセッサ１０１に通知しないよ

うにすることができるとする。
【００２３】
プロセッサ１０１は、演算装置１０３、割り込みテーブルレジスタ１０４、ページテーブルレジスタ１０５、および、アドレス変換装置１０６より構成されている。
【００２４】
割り込みテーブルレジスタ１０４は、割り込みテーブル１０７の仮想アドレスを指し示している。割り込みテーブルの詳細については後述するが、割り込み番号毎の割り込みハンドラの開始アドレスを記録している。図１で、割り込みテーブルレジスタ１０４と割り込みテーブル１０７の接続を破線で記載しているのは、割り込みテーブルレジスタ１０４が割り込みテーブルの仮想アドレスを指し示すためである。割り込みが発生すると、プロセッサ１０１は割り込み制御装置１１２から数値化された割り込み番号を受ける。この番号をインデックスとして割り込みテーブル１０７より割り込みハンドラアドレスを取得し、割り込みハンドラに制御を渡す。
【００２５】
ページテーブルレジスタ１０５は、ページテーブル１０８を指し示している。ページテーブルレジスタ１０５は、ページテーブル１０８の物理アドレスを格納している。
【００２６】
アドレス変換装置１０６は、演算装置が要求する命令アドレス、あるいは、オペランドが格納されているアドレスを受けとり、ページテーブルレジスタ１０５の指しているページテーブル１０８の内容に基づき仮想－実アドレス変換を実施する。
【００２７】
計算機１００には外部入出力装置としてキーボード１１３、ディスプレイ１１４、磁気ディスク１１５、その他の外部機器１１６、および、１１７が接続している。ディスプレイ１１４を除く機器は割り込み信号線１１０により割り込み制御装置１１２に接続している。
【００２８】
主記憶装置１０２の内容について簡単に説明する。現在、計算機１０１では２つのオペレーティングシステムが動作している。それぞれを第一のオペレーティングシステム、第二のオペレーティングシステムと呼ぶことにする。また、計算機を起動すると第一のＯＳが起動するように設定されており、外部機器１１６、および、１１７は第二のＯＳにより管理される機器であるとする。
【００２９】
第一のＯＳは、起動時の初期化処理で、その他のオペレーティングシステム用に、この場合は第二のＯＳ用に物理メモリ領域を予約する。つまり、第一のＯＳが、第二のＯＳ用に予約された物理メモリ領域を利用できないように物理メモリ領域を確保する。図１は、この奪った領域に第二のＯＳがロードされている様子を示している。
【００３０】
更に、第一のＯＳの初期化過程では、第一のＯＳから外部機器１１６、および、１１７の利用する割り込み番号や入出力アドレスを、既に利用済みであるとして予約する。
【００３１】
また、第一のＯＳは、すべてのオペレーティングシステムから参照可能な共通領域を持つ。その共通領域に、割り込みテーブル１０７、割り込み管理プログラム、割り込みハンドラ、各オペレーティングシステムから呼びだし可能なインターフェイスモジュール等を格納する。
【００３２】
本発明の実施の形態の、動作概要について説明する。本実施の形態では、第二のＯＳは第一のＯＳよりも優先して動作する。優先して動作するとは、第一のＯＳは第二のＯＳがアイドル状態であるときのみ動作可能であることを示す。第二のＯＳの処理が終了しない限り、第一のＯＳは動作できない。
【００３３】

また、第二のＯＳが管理するデバイスが割り込みを発生すると、第一のＯＳの処理は中断され、制御は第二のＯＳに移る。第二のＯＳ実行中に第一の割り込みが発生しても、その割り込み処理は第一のＯＳが実行されるまで延期される。

【００３４】

さらに、第一のＯＳと第二のＯＳは明確に区分されており、割り込みハンドラなどを配置する共通領域以外は、互いにアクセスできないようにする。これにより、２つのオペレーティングシステムが誤って互いの領域にアクセスして障害が起きることを防いでいる。

【００３５】

以下、上記の機能を実現する本発明の実施形態について説明する。

【００３６】

図２は、本発明の実施の形態における２つのオペレーティングシステムの関係を概念的に示した図である。それぞれのオペレーティングシステムは、それぞれ独立したアドレス空間を保持する。２０１は第一のＯＳの仮想空間で、２０２は第二のＯＳの仮想空間を示している。ここで、第二のＯＳの空間２０２に対応する実記憶は、図１の主記憶１０２の第二のＯＳの領域になる。

【００３７】

仮想空間の一部分には、共通領域２０３がマップされる。共通領域２０３に対応する実記憶は、図１の主記憶１０２の共通領域として示した領域である。共通領域２０３は、もともとは第一のＯＳのカーネルの領域の一部である。第二のＯＳをロードする手順が、アドレス空間２０２を構築する時に、共通領域２０３をアドレス空間２０２にマッピングするように第二のＯＳ用のページテーブルを作成する。この手順については後述する。

【００３８】

図２は、各オペレーティングシステムが管理するハードウェアを示している。第一のＯＳは、キーボード１１３、ディスプレイ１１４、および、磁気ディスク１１５を、第二のＯＳは入出力装置１１６、および、１１７を管理することを示している。クロック１１１と割り込み制御装置１１２は、もともとは第一のＯＳが管理しているハードウェアであるが、共通領域２０３中のプログラムが管理することを示している。

【００３９】

次に、ページテーブルの構成について説明する。図３は、本発明の実施の実施の形態でのページテーブルの構成を示している。

【００４０】

３００がページテーブルである。ページテーブル３００は、プロセッサ１０１の仮想アドレス空間の仮想ページ毎に、それぞれの仮想ページを記述するエントリを持っている。それぞれのエントリは、有効ビット３０１と、物理ページ番号３０２により構成される。

【００４１】

有効ビット３０１は、その仮想ページに対応する物理ページが割り当てられているか、つまり、仮想－実アドレス変換が可能かを示している。例えば、ページテーブル３００の仮想ページ３は、有効ビットがセットされていないので、仮想ページ３に対応する物理ページが存在しないことを示している。有効ビット３０１がセットされていない仮想ページへのアクセスが発生すると、プロセッサはページフォルトを発生する。

【００４２】

物理ページ番号３０２は、仮想ページに対応する物理ページ番号を記録している。

【００４３】

アドレス変換装置１０６は、ページテーブルレジスタ１０５の指し示しているページテーブルの内容を参照して、演算装置１０３の生成する仮想アドレスを実アドレスに変換する。プロセッサ１０１は、変換により得られた実アドレスにより主記憶装置１０２を参照する。

【００４４】

ページテーブルを切替えることにより独立した空間を構築することができ、図２に示した第一のオペレーティングシステムの空間、および、第二のオペレーティングシステムの空

間の構築が可能である。また、共通領域２０３については、両方のオペレーティングシステムのページテーブルの共通領域に対応する部分に、同じ物理ページをマップするように設定しておけば、共通領域を実現できる。

【００４５】

次に、割り込みテーブルの構成について説明する。図４は、割り込みテーブルの構成を示している。

【００４６】

４００が割り込みテーブルである。割り込みテーブル４００は、プロセッサ１０１が割り込み制御装置１１２から受ける割り込み番号毎の、割り込みハンドラの仮想アドレス４０１を記録している。プロセッサ１０１は割り込み要求を割り込み制御装置１１２から受けると、割り込み番号に対応する割り込みハンドラのアドレスを、割り込みテーブルレジスタ１０４の指し示している割り込みテーブル４００から取得し、そのアドレスに制御を移すことで割り込み処理を開始する。

【００４７】

図５は、割り込み制御装置１１２を示している。割り込み制御装置１１２は、割り込みマスクレジスタ５０１、および、選択装置５０２を持っている。

【００４８】

割り込みを発生する入出力装置は、割り込み信号線１１０により割り込み制御装置１１２と接続する。入出力機器の発生する割り込みは、割り込み信号線１１０のどの信号線に接続するかにより優先順位が付けられる。ここでは、割り込み０番に対応する割り込み信号がもっとも優先度が高い割り込みであるとする。

【００４９】

割り込み信号線１１０は、選択装置５０２に接続している。選択装置５０２は、割り込み信号を受けると、プロセッサがその割り込みを受け付けたことを通知するまで、未処理の割り込みがあることを記録している。

【００５０】

割り込みマスクレジスタ５０１は、入出力機器の発生した割り込みをプロセッサに通知してよいかを記録している。割り込みマスクレジスタ５０１は、プロセッサ１０１から入出力命令により設定可能である。

【００５１】

選択装置５０２は、割り込み信号線１１０から割り込み要求を受けた時と、割り込みマスクレジスタ５０１の内容が書き換えられた時に、選択装置５０２が記録している未処理割り込みと、割り込みマスクレジスタ５０２の内容を比較して、プロセッサに割り込みを通知するかどうかを決める。具体的には、選択装置５０２が記録している未処理割り込みのうち、割り込みマスクレジスタ５０１に割り込み可能と設定されていて、優先度の最も高い割り込みから順にプロセッサに通知する。選択した割り込みについて、選択装置５０２は、通知する割り込み信号に対応する数字信号を、割り込みバス１１９経由でプロセッサ１０１に送る。

【００５２】

プロセッサ１０１は、割り込みを受けた時に、入出力命令により選択装置５０２に記録されている未処理割り込み記録を解消できる。

【００５３】

次に、本発明の実施の形態における、計算機のブート手順について説明する。

【００５４】

ブート手順の始めの部分は、読みとり専用記憶装置である１１８に格納されている。記憶装置１１８は、プロセッサの物理アドレス空間のある決められたアドレスにマップされるようにバス１０９を介してプロセッサ１０１に接続している。この手順は、ハードウェア構成の検出、オペレーティングシステムカーネルをロードするプログラムの主記憶へのローディングを実施する。

【００５５】

プロセッサ１０１がリセットされると、プロセッサ１０１は予め定められた物理アドレスに制御を移す。記憶装置１１８は、この時に実行されるプログラムを格納しており、プロセッサ１０１がリセットされた時にこのプログラムに制御を渡せるように物理アドレス空間にマップされている。

【００５６】

記憶装置１１８に格納されているプログラムは、磁気ディスク装置１１２に格納されている第一のＯＳのカーネルローダを主記憶装置１０２にロードして実行する。カーネルローダは、磁気ディスク装置１１２の予め定められた位置にあり、記憶装置１１８に格納されたプログラムは、容易にこれを見つけることができる。

【００５７】

カーネルローダの処理手順について説明する。図６は、本発明の実施の形態における、オペレーティングシステムのカーネルローダの処理手順を示すフローチャートである。

【００５８】

このカーネルローダは、オペレーティングシステムのファイルシステム構造を理解して、ファイル名よりファイルの格納位置を特定し、主記憶に読み込むことができるよう構成されている。

【００５９】

カーネルローダの処理手順について説明する。まず、カーネルにパラメータとして渡す主記憶リスト１１０１、ロードモジュールリスト１１０４、および、デバイスリスト１１０２を初期化し、カーネル用のページテーブル領域を割り当てる（ステップ６０１）。３つのリストの構成については後述する。

【００６０】

主記憶リスト１１０１は、主記憶装置１０２の利用状況を示すデータ構造であり、カーネルローダが以降の処理で物理メモリの割り当てをする場合は、主記憶リスト１１０１を参照、および、変更して実施する。

【００６１】

次に、ハードウェア構成の検査（ステップ６０２）、および、ハードウェア構成データの作成（ステップ６０３）を実施する。ステップ６０２においては、計算機１００にどのような接続されているかハードウェアが検査する。続くステップ６０３では、ステップ６０２の結果に基づいてハードウェア構成に関するデータ構造であるデバイスリスト１１０２を作成する。オペレーティングシステムカーネルは、このデバイスリスト１１０２を参照してカーネル初期化処理を実施する。

【００６２】

次に、オペレーティングシステムカーネルの構成情報７００を磁気ディスク装置１１２より読み込み、パラメータテーブル１１００に構成情報のアドレスを設定する（ステップ６０４）。オペレーティングシステムのカーネルは、カーネル本体のファイルと、その他のデバイスドライバのファイルといったように、複数のファイルから構成されていても良い。構成情報７００は、予め決められたファイル名で磁気ディスク１１２に格納されており、ロードプログラムはこれを見つけることができる。

【００６３】

本発明の実施形態におけるカーネル構成情報のデータ構造を図７に示す。７００は、カーネル構成情報を記録しているファイルの内容を示している。構成情報ファイル７００は、カーネルローダやオペレーティングシステムが参照するデータを格納している。格納されているデータには名前がつけられており、プログラムは名前からそれに対応するデータを取得することができる。図７の例では、名前がオブジェクトファイル（７０１）というエントリがあり、そのデータが７０２に格納されている。また、ｓｅｃｏｎｄａｒｙ　ＯＳには、第二のＯＳ用のデータ（７０４）を格納しているとする。

【００６４】

カーネルローダの処理手順の説明を続ける。構成情報７００を読み込んだ後、構成情報７００中のオブジェクトファイルという名前のつけられたデータに格納されているカーネル

構成ファイルのすべてについて、主記憶装置１０２に読み込み（ステップ６０６）、ロードモジュールリスト１１０４にエントリを追加し（ステップ６０７）、カーネル用のページテーブルの設定（ステップ６０８）を実施する。ここでは、ｋｅｒｎｅｌ、ｄｒｉｖｅｒ１、および、ｄｒｉｖｅｒ２というファイル名のオブジェクトファイルをロードする。

【００６５】

ロードモジュールリストエントリの追加と、カーネル用のページテーブルの設定は、主記憶１０２にロードしたオブジェクトファイルに格納されているデータにしたがって実施する。カーネルを構成するオブジェクトファイルには、そのファイル内容をマップする仮想アドレス、ファイルの大きさなどが含まれている。これを参照してページテーブルを構築する。オブジェクトファイルのデータ構造については後述する。

【００６６】

最後に、ページテーブルレジスタ１０５を、構築したページテーブルのアドレスに設定し、プロセッサを仮想アドレス変換モードに移行させ（ステップ６０９）、構築した主記憶リスト１１０１、デバイスリスト１１０２、カーネル構成情報テーブル１１０３、および、ロードオブジェクトリスト１１０４の組から成るパラメータテーブル１１１０をパラメータとして、カーネルの初期化ルーチンに制御を渡す（ステップ６１０）。カーネルのエントリポイントは、カーネルファイル内のデータに記録されている。

【００６７】

次に、カーネルを構成するオブジェクトファイルの構造について説明する。図８は、本発明の実施の形態での、カーネルを構成するオブジェクトファイルの構造を示す図である。

【００６８】

８００は、オブジェクトファイル全体を示している。オブジェクトファイル８００は、８０１ないし８１１のヘッダ部分と、８１２ないし８１３のセクション部分より構成される。

【００６９】

ヘッダ部分の構成について説明する。ヘッダマップアドレス８０１とヘッダサイズ８０２は、オブジェクトファイル８００ヘッダ部分のカーネル空間での格納位置を記述している。ヘッダ部分は、ヘッダマップアドレス８０１に記録されているアドレスに読み込まれる。

【００７０】

初期化エントリ８０３は、オブジェクトファイル８００の初期化用ルーチンのアドレスを記録している。カーネルは、カーネル初期化時に各オブジェクトファイルの初期化ルーチンを呼ぶときに、各オブジェクトファイルの初期化エントリ８０３を参照して初期化ルーチンを見つける。

【００７１】

セクション数８０４は、オブジェクトファイル８００に含まれているセクションの数を記録している。セクションとは、オブジェクトファイル内の連続しているデータ領域で、これを単位としてカーネルの仮想空間へのマッピングを決定する。例えば、オブジェクトファイルは、実行コードが格納されているセクションと、そのオブジェクトファイルが参照するデータを格納しているセクションを含んでいる。これらのセクションはオブジェクトファイル作成時にコンパイラにより作成される。

【００７２】

外部参照テーブルオフセット８０５と外部参照テーブルサイズ８０６は、このオブジェクトファイル内の実行コードが参照する、他のオブジェクトファイルの公開参照の情報を格納する外部参照テーブル８１０を記述している。外部参照テーブル８１０は、オブジェクトファイル８００のヘッダ部分に含まれており、外部参照テーブルオフセット８０５は、ヘッダの先頭からの外部参照テーブル８１０のオフセットを記録している。

【００７３】

公開参照テーブルオフセット８０７と公開参照テーブルサイズ８０８は、このオブジェクトファイルが他のオブジェクトファイルの実行コードに公開しているモジュールとデータ

の情報を格納している公開参照テーブル８１１を記述している。公開参照テーブル８１１は、オブジェクトファイル８００のヘッダ部分に含まれており、公開参照テーブルオフセット８０７は、ヘッダの先頭からの公開参照テーブル８１１のオフセットを記録している。

【００７４】

セクションデータ８０９は、オブジェクトファイル８００に含まれる各セクションについてのデータを格納している。セクションデータは、セクション数８０４の数だけある。セクションデータの構成については後述する。

【００７５】

セクションデータの後に外部参照テーブル８１０と、公開参照テーブル８１１が続き、ヘッダ部分を構成する。

【００７６】

ヘッダ部分の後には、各セクションの本体８１２、８１３が格納されている。

【００７７】

セクションデータ８０９の構成について説明する。セクション開始オフセット８２０とセクションサイズ８２１は、オブジェクトファイル８００内での、当該セクションの開始オフセットと、当該セクションの大きさを記録している。

【００７８】

セクションは、セクションマップアドレス８２２に記録されたアドレスに配置されるようにカーネルの仮想空間にマップされる。セクション名称８２３には、当該セクションの名前を示す文字列が格納されている。

【００７９】

外部参照テーブルの構造について説明する。図９は、外部参照テーブルの構造を示している。テーブル８１０の先頭には、このテーブルに含まれる外部参照情報の数９０１が格納されている。

【００８０】

続いて、オブジェクトファイル名９０２、外部参照名９０３が格納されている。オブジェクトファイル名９０２と外部参照名９０３は、文字列テーブル９０５へのオフセット値を格納しており、実際の文字列による名称は文字列テーブル９０５内に格納されている。

【００８１】

外部参照アドレス９０４には、この当該外部参照エントリで記述される外部参照の実際のアドレスが格納される。カーネルは、オブジェクトファイルを主記憶にロードするときに、当該の外部参照を含むオブジェクトファイルの公開参照のテーブルを参照して関数、あるいは、データのアドレスを取得し、外部参照アドレス９０４に設定する。オブジェクトファイルの実行コードは、外部関数アドレス９０４に格納されたアドレスを参照して、他のオブジェクトファイル内の関数の呼び出しや、データの参照をするようコンパイルされており、他オブジェクトモジュールにある関数の実行や、データ参照が可能である。

【００８２】

オブジェクトファイル名９０２、外部参照名９０３、および、外部参照アドレス９０４が１つの外部参照を定義し、外部参照数９０１に記録されている数だけ、これらのエントリが連続して配置される。その後に、文字列テーブル９０５が格納される。文字列テーブルは、オブジェクトファイル名や、外部参照名の文字列を格納している。

【００８３】

公開参照テーブルの構造について説明する。図１０は、公開参照テーブルの構造を示す図である。

【００８４】

テーブル８１１の先頭には、この公開参照テーブル８１１により他のオブジェクトモジュールに公開される参照名の数１００１が記録されている。１つの公開参照は、公開参照名１００２と公開参照アドレス１００３により記述される。公開参照名１００２は、文字列テーブル１００４へのオフセット値を格納しており、実際の文字列による名前は文字列テ

ーブル１００４に格納されている。公開参照アドレス１００３は、この参照に対応するアドレスを格納している。

【００８５】

次に、ステップ６０１から始まるブート手順が作成するハードウェア構成データと、ロードオブジェクトデータの構成について説明する。図１１がハードウェア構成データとロードオブジェクトデータの構成を示す図である。

【００８６】

パラメータテーブル１１００は、カーネルローダが作成するデータ構造である。パラメータテーブル１１００から始まる３つのリストは、ローダが構築するカーネルの仮想空間に配置されるので、カーネルから参照可能である。

【００８７】

パラメータテーブル１１００は、ローダが構築した３つのリストの先頭へのポインタと、１つのテーブルへのポインタを保持している。３つのリストとは、主記憶リスト１１０１、デバイスリスト１１０２、および、ロードオブジェクトリスト１１０４で、１つのテーブルはカーネル構成情報テーブル１１０３である。それぞれについて説明する。

【００８８】

主記憶リスト１１０１は、主記憶ブロック記述データ１１１０のリストである。主記憶ブロック記述データ１１１０は、ベースアドレス１１１１、ブロックサイズ１１１２、ブロック利用状況１１１３、および、次の主記憶ブロック記述データへのポインタ１１１４から構成されている。

【００８９】

主記憶ブロック記述データは、ある連続した主記憶領域についての利用状況を記録している。ベースアドレス１１１１は連続領域の開始物理アドレスを示し、ブロックサイズ１１１２は連続領域の大きさを格納している。ブロック利用状況１１１３は、当該連続領域が未使用であるか、あるいは、ローダにより割り当て済みであるかを示す値が格納されている。そして、次エントリへのポインタ１１１４によりリストを構成している。図１１では、１１１０の次のエントリは１１２０である。主記憶リスト１１０１を参照することで、物理メモリの利用状態を知ることができる。

【００９０】

デバイスリスト１１０２は、カーネルローダが検出したハードウェアデバイスに関するデータを格納しており、ステップ６０３で作成されている。デバイスリスト１１０３は、デバイスデータ１１５０からなるリストである。デバイスデータ１１５０は、デバイスタイプ１１５１、デバイス情報１１５２、および、次のデバイスデータへのポインタ１１５３より構成される。

【００９１】

デバイスタイプ１１５１は、このデバイスデータエントリ１１５０により記述されるデバイスの種類を示す値が格納されている。デバイス情報１１５２は、デバイスの種類に特有なデータを格納している。例えば、割り込み番号やＩ／Ｏアドレスなどがそれに相当する。そして、次エントリへのポインタ１１５３によりリストを構成している。

【００９２】

カーネル構成情報テーブルへのポインタ１１０３は、カーネルローダが主記憶１０２に読み込んだカーネル構成情報ファイル７００の内容を指し示している。

【００９３】

ロードオブジェクトリスト１１０４は、カーネルローダが主記憶にロードしたオブジェクトファイルに関するデータを保持している。ロードオブジェクトリストは、ロードオブジェクトデータ１１３０のリストである。ロードオブジェクトデータ１１３０は、オブジェクトファイル名１１３１、オブジェクトアドレス１１３２、および、次のロードオブジェクトデータへのポインタ１１３３より構成されている。

【００９４】

オブジェクトファイル名１１３１は、ロードオブジェクトデータ１１３０により記述され

ているオブジェクトファイルのファイル名である。オブジェクトアドレス１１３２は、当該オブジェクトファイルのヘッダ領域がロードされているカーネル空間のアドレスを格納している。そして、次エントリへのポインタ１１３３によりリストを構成している。

【００９５】

ロードオブジェクトリスト１１０４は、カーネルローダがカーネルを構成するオブジェクトファイルを読み込む時に同時に作成している（ステップ６０７）。

【００９６】

次に、本発明の実施の形態における第一のＯＳの初期化手順について説明する。図１２は、第一のＯＳの初期化手順を示すフローチャートである。

【００９７】

まず、パラメータとして渡されたパラメータテーブル１１００中のロードオブジェクトリスト１１０４を参照して、カーネルローダがロードしたオブジェクトファイルの外部参照アドレス解決を実施する（ステップ１２０１）。アドレス解決では、各オブジェクトファイルにある外部参照テーブル８１０の、外部参照アドレス９０４を決定する。アドレスは、各オブジェクトファイルの公開参照テーブル８１１を参照して決定する。

【００９８】

続く、ステップ１２０２では、カーネル起動時のパラメータとして渡されたパラメータテーブル１１００の主記憶リスト１１０１を参照して、第二のＯＳ用に主記憶領域を確保する。

【００９９】

具体的には、カーネル構成情報テーブル７００より第二のＯＳの情報を取り出す。図７の例では、第二のＯＳの構成情報は７０４に格納されている。この構成情報７０４を参照して、確保すべき主記憶の大きさを決定する。そして、主記憶リスト１１０１の空きブロックエントリの内容を変更して、主記憶領域を割り当てる。この処理は、第一のＯＳが空きメモリ管理を始める前に実施する。

【０１００】

これにより、第一のＯＳから見ると、第二のＯＳに割り当てた主記憶領域は存在しないことになり、第一のＯＳから参照されることがなくなる。したがって、割り当てた領域は、第二のＯＳが自由に使うことのできる主記憶領域となる。これは、図１の第二のＯＳの領域に相当する。

【０１０１】

次のステップ１２０３では、カーネル内部のデータ構造の初期化を実施する。この初期化には、後で述べるデバイス管理テーブルの初期化も含む。

【０１０２】

ステップ１２０４では、第二のＯＳが管理するデバイスを予約する。ここで予約するとは、第一のＯＳから利用できないようにすることである。具体的には、第一のＯＳが管理しているデバイス管理テーブルへの登録を実施する。

【０１０３】

第二のＯＳが管理するデバイス資源は、パラメータテーブル１１００のカーネル構成情報テーブル１１０３の指すテーブル７００に格納されている、第二のＯＳの構成情報を参照して決める。この実施形態では、図７の７０４に格納されているデータがそれに相当する。

【０１０４】

デバイス管理テーブルについて説明する。図１３は、第一のＯＳのデバイス管理テーブルの構造を示した図である。デバイス管理テーブルは、割り込みベクタ管理テーブル１３００と、Ｉ／Ｏアドレス管理リスト１３１０の２つのデータ構造からなる。

【０１０５】

割り込みベクタ管理テーブル１３００は、プロセッサ１０１が受け付ける各割り込み番号について、第一のＯＳがその割り込み番号を利用するかどうかを示す値を格納している。カーネルは、デバイスドライバが初期化時に割り込み番号を要求した場合に、このテーブ

ル１３００を検査し、要求された割り込み番号が利用されているか検査し、そうでない場合にのみ要求された割り込み番号を使用する権利をデバイスドライバに与える。既に利用済みであると記されている場合は、そのデバイスは第一のＯＳからは利用できないことになる。

【０１０６】

図２の入出力装置１１６と１１７を例として説明する。入出力装置１１６と１１７は、それぞれ割り込み番号４と５と要求すると仮定する。入出力装置１１６と１１７は、第二のＯＳが管理するデバイスである。入出力装置１１６と１１７の要求する割り込み番号は、カーネル構成情報テーブル７００の第二のＯＳの構成情報７０４に記録されている。ステップ１２０４では、この構成情報７０４を参照して、割り込みベクタ管理テーブルの、割り込み番号４と５のエントリに利用中であることを示す値を格納する。この処理は、第一のＯＳがデバイス管理を開始する前に実施するため、第一のＯＳは、入出力装置１１６と１１７にアクセスすることができなくなり、装置１１６と１１７を第二のＯＳの管理下におくことができる。

【０１０７】

Ｉ／Ｏアドレス管理リスト１３１０についても同様である。Ｉ／Ｏアドレス管理リスト１３１０は、Ｉ／Ｏアドレス範囲を表現するエントリ１３２０からなるリストである。エントリ１３２０は、第一のＯＳが利用するＩ／Ｏアドレス範囲１３２１と、リストを構成するための次のエントリへのポインタ１３２２からなる。割り込みベクタ管理テーブル１３００と同様、デバイスドライバが初期化時にＩ／Ｏアドレス範囲を要求した場合、カーネルは、そのアドレス範囲が既に利用されているかＩ／Ｏアドレス管理リスト１３１０により検査し、未使用である場合、このリスト１３１０にエントリを追加して、利用許可を与える。

【０１０８】

第二のＯＳが管理するデバイスが要求するＩ／Ｏアドレス範囲は、割り込み番号と同様にカーネル構成情報テーブル７００に格納されているので、それを参照すれば要求アドレスを知ることができ、第一のＯＳがデバイス管理を開始する前にＩ／Ｏアドレスを予約できる。

【０１０９】

ステップ１２０２の処理により、第一のＯＳから完全に独立した第二のＯＳ専用の空間を構築することが可能になる。さらに、ステップ１２０４の処理により、第一のＯＳ上で動作するユーザプログラムは、第二のオペレーティングシステムが管理するデバイス、この例では、入出力装置１１６と１１７へのアクセスが不可能になる。また、装置１１６と１１７の割り込み番号とＩ／Ｏアドレスを利用するデバイスドライバを導入することを禁止できる。

【０１１０】

この２つのステップの処理の効果として、第一のＯＳが関知しない部分に第二のＯＳを導入することが可能になる。

【０１１１】

続くステップ１２０５、ないし、ステップ１２０７は通常のオペレーティングシステムの初期化処理と同じである。ステップ１２０５のシステムデバイスの初期化では、カーネルが直接管理するシステムデバイスの初期化を実施する。システムデバイスとは、クロック割り込みなど、第一のＯＳの実行に不可欠で、第一のＯＳが必ず存在していると仮定しているデバイスである。

【０１１２】

ステップ１２０６では、カーネルローダがロードしたオブジェクトファイルについて、それぞれの初期化エントリを実行する。初期化エントリアドレスは、オブジェクトファイルのヘッダ部分に格納されている。最後に、初期プロセスを作成する（ステップ１２０７）。

【０１１３】

次に、本発明の実施の形態における、第二のＯＳのロード手順について説明する。図１４は、第二のＯＳのロード手順を示すフローチャートである。

【０１１４】

まず、第二のＯＳ用に割り当てた物理メモリ領域に、第二のＯＳのオブジェクトファイルを読み込む必要がある。しかし、第二のＯＳの物理メモリ領域は、そのままでは第一のＯＳから書き込むことはできないので、割り当てた物理メモリ領域を第一のＯＳの仮想空間に一時的にマッピングする（ステップ１４０１）。

【０１１５】

ステップ１４０２では、マッピングした領域に、第一のＯＳのファイル読み込み手順を利用して、第二のＯＳのオブジェクトファイルを読み込む。なお、第二のＯＳのオブジェクトファイルの形式は、第一のＯＳのオブジェクトファイル形式８００と同じ形式であるとする。

【０１１６】

次に、第二のＯＳ用のページテーブルを作成する（ステップ１４０３）。このページテーブルも第二のＯＳ用の領域に作成する。この時に、第一のＯＳと共有する部分について、第二のＯＳの空間からも参照できるように、ページテーブルを構築する。

【０１１７】

共通領域２０３について、割り込み処理や共通データの管理を実施するデバイスドライバ（以下サポートドライバ）をロードした領域を共通領域２０３とする。このデバイスドライバがロードされたアドレスは、ロードオブジェクトリスト１１０４より知ることができる。また、

続くステップ１４０４で、第二のＯＳのカーネルの外部参照を解決する。但し、第二のＯＳが直接参照できる他のオブジェクトファイルの参照は、共通領域２０３に配置されている関数とデータ、つまり、サポートドライバの公開参照のみである。したがって、ここでは、サポートドライバのオブジェクトファイルのヘッダ部分に格納されている公開参照テーブル８１１を参照して、第二のＯＳのカーネルオブジェクトファイルの外部参照テーブル８１０の外部アドレス９０４を決定する。

【０１１８】

次に、第二のＯＳの公開参照のアドレスを、共通領域のデータ領域に割り当てられた外部参照アドレステーブルに書き込む。共通領域となっているサポートドライバは、第一のＯＳのデバイスドライバとして、第一のＯＳの機構にしたがって読み込むため、第二のＯＳの公開参照とリンクすることはできない。

【０１１９】

ここでは、サポートドライバのデータ領域内に必要な外部参照名と、それに対応する外部アドレスを格納するテーブルを予め用意しておく。サポートドライバの実行コードは、このテーブルを参照して第二のＯＳのカーネルの公開関数の呼びだし、公開データの参照を実施するように記述する。そして、第二のＯＳのロード時に、このテーブルの外部アドレス欄にサポートドライバの公開参照のアドレスを書き込むこととする。

【０１２０】

これで、第二のＯＳ領域の設定を終了し、第一のＯＳのカーネル領域にマップした、第二のＯＳ用の物理メモリ領域のマッピングを解除する（ステップ１４０６）。

【０１２１】

次に、ＯＳコンテクストテーブル１５１０の第二のＯＳのコンテクストと、ＯＳ識別変数１５３０を設定する（ステップ１４０７）。ＯＳコンテクストは、実行オペレーティングシステムを切り替えるときに参照するデータ構造で、ページテーブルアドレス値とスタックポインタの初期値とで構成する。ここでは、ページテーブルレジスタ値として第二のＯＳをマップするページテーブルのアドレスを、スタックポインタ値として第二のＯＳのカーネルスタックの初期アドレスを設定する。ＯＳ識別変数１５３０には、第一のＯＳが実行中であることを示す値を格納する。ＯＳコンテクストテーブル１５１０とＯＳ識別変数１５３０については後述する。

【０１２２】
次に、第二のＯＳの初期化モジュールを実行する（ステップ１４０８）。これには、オペレーティングシステム空間の切替が伴う。オペレーティングシステムの切替えについては、別のフローチャートにより説明する。また、第二のＯＳの初期化モジュールは公開参照になっており、サポートドライバはそのアドレスを知ることができる。
【０１２３】
最後に、ステップ１４０９にて、現在の割り込みテーブル１０４に登録されている第一のＯＳの割り込みハンドラのアドレスを、割り込み識別テーブル１５２０のハンドラの欄１５２２にコピーし、割り込みテーブルレジスタ値を、サポートドライバに割り当てた割り込みテーブルのアドレスに変更する。これは、プロセッサ１０１の割り込みテーブルレジスタ１０４の変更により実施する。
【０１２４】
割り込みテーブルをサポートドライバ内のテーブルに変更するのは、割り込み発生時にどちらのオペレーティングシステムが実行していても、常にプロセッサ１０１の仮想アドレス空間に割り込みテーブルが存在している必要があるためである。割り込みテーブルに登録される割り込みハンドラも、サポートドライバ内に配置する。サポートドライバの領域は、ステップ１４０３にて、第二のＯＳの仮想空間にもマッピングして共通領域２０３とするので、いつでも参照できることになる。サポートドライバの割り込み処理については後述する。
【０１２５】
また、ステップ１４０９では、第一のＯＳの割り込み管理情報も変更する。具体的には、割り込み禁止レベルに関連するデータ構造を変更するが、これについては後述する。
【０１２６】
共通領域２０３に格納するデータ構造について説明する。図１５は、共通領域２０３のうちのデータ領域１５００に格納するデータ構造を示した図である。図１５にしたがって順に説明する。
【０１２７】
１５１０は、ＯＳコンテクストテーブルである。ＯＳコンテクストテーブル１５１０は、第一のＯＳと第二のＯＳとの間の切替えに必要なデータを保持する。この実施の形態では、第一のＯＳは第二のＯＳがアイドル状態の時のみ走行できるとする。この場合、第一のＯＳ実行中のある時点で第二のＯＳへの切替が起こり、第二のＯＳの実行が終了した時点で、第一のＯＳに制御を戻せば良い。
【０１２８】
したがって、それぞれで保存しておかなければならないコンテクストは１組で良い。第一のＯＳのコンテクストについては、ＯＳ切替が要求された時点でのページテーブルレジスタ値１５１１と、スタックポインタ値１５１２を保存しておけば、第二のＯＳ実行終了後に、第一のＯＳに制御を復帰させることができる。
【０１２９】
また、第一のＯＳから第二のＯＳへ制御を切り替えるときには第二のＯＳは動作していない。したがって、第二のＯＳのコンテクストは、ページテーブルアドレスもスタックポインタも固定の値でよい。第二のＯＳのページテーブルレジスタ値１５１３とスタックポインタ値１５１４は、第二のＯＳをロードするときに設定する（ステップ１４０７）。
【０１３０】
１５２０は、割り込み識別テーブルである。割り込み識別テーブル１５２０は、外部割り込みの割り込み番号毎に、どちらのオペレーティングシステムが割り込みを処理するかを示す値１５２１と、割り込みハンドラのアドレス１５２２が記録されている。外部割込みが発生すると、共通領域２０３内の割り込みハンドラが割り込みを捕獲し、この割り込み識別テーブル１５２０の処理ＯＳ１５２１を参照して、どちらのＯＳに処理させるかを決定し、ハンドラ１５２２のアドレスへ制御を渡す。割り込み処理の詳細については、後述する。

【０１３１】
１５３０は、実行中のオペレーティングシステムを示す値を格納しているＯＳ識別変数である。この変数１５３０は、ステップ１６０１から始まるＯＳ切り替え手順でＯＳ切り替えの度に設定する。割り込み処理では、この変数１５３０を参照して割り込み処理手順を決定する。
【０１３２】
１５４０は、第二のＯＳの実行中に、第一のＯＳが管理しているデバイスの割り込みが発生したかを示す遅延割り込み状態変数である。この変数１５４０は、どの割り込み番号の割り込みが発生したかを記録している。ＯＳ切り替え手順は、第二のＯＳの実行が終了したときにこの変数１５４０を検査して、割り込み処理を起動するか決定する（ステップ１６０８）。
【０１３３】
オペレーティングシステムの切替え手順について説明する。図１６は、本発明の実施の形態における、オペレーティングシステムの切替え手順を示すフローチャートである。この切り替え手順は、第一のＯＳの実行中に呼び出され、第二のＯＳの切り替えを実施する。
【０１３４】
図１６に示した手順は、第二のＯＳへの切り替え後に実行する第二のＯＳのモジュールのアドレスと、そのモジュールへ渡す引数を引数として受ける。第二のＯＳのモジュールのアドレスは、共通領域２０３内に設定した、外部参照アドレステーブルを参照すれば知ることができる。
【０１３５】
まず、始めのステップ１６０１で、現在のスタックポインタ値とページテーブルレジスタ値を、ＯＳコンテクストテーブル１５１０の、第一のＯＳのコンテクストとして保存する。ステップ１６０１では、現在のスタックポインタ値を１５１２に、現在のページテーブルレジスタ１０５の値を１５１１に保存する。
【０１３６】
他のレジスタコンテクストについて、ＯＳコンテクストテーブル１５１０に保存する必要はない。必要があれば、第一のＯＳのスタックに保存すればよい。
【０１３７】
スタックポインタとページテーブルレジスタ値を保存した後、ステップ１６０２にて、ページテーブルレジスタ１０５に第二のＯＳを仮想空間にマップするページテーブルのアドレスを設定する。これは、ＯＳコンテクストテーブル１５１０の１５１３に記録されている。更に、スタックポインタを第二のＯＳ用に設定する。これも、テーブル１６００の第二のＯＳのスタックポインタ１５１４に格納されている。
【０１３８】
次のステップ１６０３で、第一のＯＳの割り込み状態を示す遅延割り込み状態変数１５４０をクリアする。状態変数１５４０は、第二のＯＳ実行中に発生した、第一のＯＳが管理しているデバイスからの割り込みの発生状況を記録する変数である。第二のＯＳを実行する前に、これをクリアしておく。
【０１３９】
そして、現在実行中のＯＳの示しているＯＳ識別変数１５３０を、第二のＯＳを示す値に書き換える（ステップ１６０４）。スタックポインタ、ページテーブルレジスタ１０５、および、ＯＳ識別変数１５３０は、常に一貫した値になっていなければならないので、ここまでのステップ１６０１、ないし、１６０４は全ての外部割り込みを禁止した状態で実行しなければならない。
【０１４０】
続くステップ１６０５で、引数として渡されたモジュールのアドレスへ制御を移し、第二のオペレーティングシステムに制御を渡す。本発明のこの実施の形態においては、第一のＯＳは第二のＯＳが実行していない時、つまり、第二のＯＳがアイドル状態のときだけ実行できるとする。したがって、第二のＯＳの処理が終了した時に、ステップ１６０６へ制

御が戻る。
【０１４１】
ステップ１６０６では、ステップ１６０１でＯＳコンテクストテーブル１５１００に保存したページテーブルレジスタ値１５１１と、スタックポインタ値１５１２のそれぞれを回復する。続くステップ１６０７で、ＯＳ識別変数１５３０を第一のＯＳが実行中であることを示す値に変更する。この２つのステップの処理も割り込みを禁止した状態で実行しなければならない。
【０１４２】
次に、第二のＯＳの実行中に発生した、第一のＯＳが管理するデバイスの外部割り込みを処理する。まず、ステップ１６０８では、遅延割り込み状態変数１５４０を検査して、割り込みが発生したかどうか検査する。発生していない場合は、ＯＳ切替え手順は終了し、呼出元に復帰する。
【０１４３】
そうでない場合、割り込みが発生している場合は、ステップ１６０９を実行する。このステップでは、第二のＯＳの実行中に発生した割り込みを、第一のＯＳが管理している延期割り込み状態変数に、未処理の割り込みがある旨を記録する。続いて、第一のＯＳの割り込み処理を起動する（ステップ１６１０）。全ての割り込み処理が終了した時に、ＯＳ切替え手順の呼出元に復帰する。
【０１４４】
本発明の実施の形態における割り込み処理について説明する。図１７は、本実施形態の割り込み処理手順を示すフローチャートである。この手順を実行するモジュールは、割り込みハンドラとしてプロセッサの割り込みテーブル１０７に登録される。さらに、この割り込みハンドラは、両方のオペレーティングシステムから参照できる共通領域２０３に配置する。
【０１４５】
外部割り込みが発生して、プロセッサ１０１により割り込みハンドラが起動されると、割り込みハンドラは割り込み要因を検査し、割り込みを発生したデバイスが第一のＯＳが管理するデバイスか、第二のＯＳが管理するデバイスか判定する（ステップ１７０１）。この判定は、割り込み識別テーブル１５２０を割り込み番号をインデックスとしてＯＳ欄１５２１を参照することより実施する。第一のＯＳのデバイスである場合はステップ１７０２へ、第二のＯＳのデバイスの場合はステップ１７０５へ進む。例えば、図１５でいえば、割り込み番号が１であれば第一のＯＳの割り込みであり、割り込み番号４であれば第二のＯＳの割り込みとなる。
【０１４６】
割り込みが第一のＯＳのデバイスの割り込みである場合、ステップ１７０２を実行する。ステップ１７０２では、割り込み発生時に実行していたＯＳを判定する。この判定は、ＯＳ識別変数１５３０を参照して実施する。実行中のＯＳが第一のＯＳの場合はステップ１７０３へ、第二のＯＳの場合はステップ１７０４へ進む。
【０１４７】
ステップ１７０３から始まる処理は、第一のＯＳが管理しているデバイスが、第一のＯＳを実行中に割り込みを発生した場合の処理である。ステップ１７０３では、あたかもステップ１７０１から始まる処理が存在せず、第一のＯＳの割り込みハンドラが、直接プロセッサ１０１から制御を受けたように見えるようにコンテクストを設定する。ここでコンテクストとは、スタックの内容やレジスタの内容を示す。そして、第一のＯＳの割り込みハンドラへ制御を渡す。第一のＯＳの割り込みハンドラのアドレスは、割り込み識別テーブル１５２０のハンドラ欄１５２２に格納されている。例えば、割り込み番号１の割り込みであるならば、１をインデックスとして割り込み識別テーブルを参照して、ハンドラアドレスを求める。
【０１４８】
この場合、ステップ１７０１から始まる手順には制御は戻らず、第一のＯＳが処理を続け

る。
【０１４９】
第一のＯＳが管理しているデバイスが、第二のＯＳを実行中に割り込みを発生した場合、ステップ１７０４を実行する。ステップ１７０４では、遅延割り込み状態変数１５４０に割り込みを発生したデバイスの割り込み番号を記録する。割り込みハンドラの処理はこれで終了する。この場合の割り込みの処理は、実行ＯＳが第一のＯＳに切り替わったときに実行される（ステップ１６０８）。
【０１５０】
発生した外部割込みが、第二のＯＳが管理するデバイスの割り込みだった場合、ステップ１７０５へ進み、どちらのＯＳが実行中であるか検査する。ここでも、ＯＳ識別変数１５３０によって実行中のＯＳを判定する。第一のＯＳが実行中の場合は、ステップ１７０６へ、第二のＯＳが実行中の場合はステップ１７１１へ進む。
【０１５１】
第二のＯＳが管理するデバイスの割り込みが、第二のＯＳの実行中に発生した場合、ステップ１７１１を実行する。ステップ１７１１は、第二のＯＳの割り込みハンドラを起動する。第二のＯＳの割り込みハンドラのアドレスは、割り込み識別テーブル１５２０のハンドラ欄１５２２に記録されている。第二のＯＳの割り込みハンドラ処理が終了して制御が戻ってきたら、この割り込みハンドラも終了し、割り込まれた時のコンテクストを回復して制御を元に戻す。
【０１５２】
第二のＯＳが管理するデバイスの外部割込みが、第一のＯＳの実行中に発生した場合、ステップ１７０６を実行する。この場合は、第一のＯＳの実行よりも第二のＯＳの処理を優先して実行する。
【０１５３】
まず、ステップ１７０６では、コンテクストを保存する。ここでのコンテクストとは、割り込み処理が終了した後で第一のＯＳに戻すときに、割り込まれたときの状態を回復するのに必要なスタックの内容とレジスタの内容を示す。このコンテクストは、第一のＯＳのカーネルのスタックに保存する。
【０１５４】
続いて、実行ＯＳの切り替えと第二のＯＳの割り込み処理の起動を実行する（ステップ１７０７、１７０８）。これは、ステップ１６０１から始まる手順により実行する。
【０１５５】
第二のＯＳの処理が終了した時点で、第一のＯＳへの切り替えを実行し（ステップ１７０９）、割り込み時のコンテクストを回復し（ステップ１７１０）、第一のＯＳの処理を再開する。ステップ１７０９の処理は、必ずしも、ステップ１７０１から始まる処理と同一のモジュール内で実行されなくてもよい。第一のＯＳへの切り替えにより処理はこのモジュールへ復帰する。
【０１５６】
２つのオペレーティングシステムで共有しているクロック割り込みの処理について説明する。クロック割り込みはは、共通領域内の割り込みハンドラにより捕獲する。この割り込みハンドラでは、まず、第二のＯＳのクロック割り込み用の割り込みハンドラを実行する。第二のＯＳの割り込みハンドラはハンドラ２欄１５２３に格納されている。第二のＯＳの割り込みハンドラの実行が終了したら、図１７のステップ１７０２から始まる処理により第一のＯＳの割り込み処理を実行する。第一の割り込みハンドラのアドレスはハンドラ欄１５２２に格納されている。
【０１５７】
次に、第一のＯＳの割り込み制御部分について説明する。これは、第一のＯＳの割り込み制御によって、誤って第二のＯＳが管理するデバイスの割り込みが禁止にされてしまわないようにするための処理である。
【０１５８】

第一のＯＳは、割り込み禁止レベルにより割り込みを制御しているとする。割り込み禁止レベルは、オペレーティングシステムのカーネル内の割り込み処理の延長で動作する部分と、そうでない部分との間の排他制御を実現するために必要となる機構である。

【０１５９】

第一のＯＳは、割り込み制御装置１１２をプログラムすることで割り込み禁止レベルを実現する。つまり、割り込み制御装置１１２の割り込みマスクレジスタ５０２をプログラムして、選択的に外部割り込みをマスクにする。第一のＯＳは第二のＯＳについて全く知らないので、第一のＯＳが割り込み禁止レベルを変更したときに、第二のＯＳのデバイスの割り込みがマスクされてしまう可能性がある。これを防ぐために、第一のＯＳの割り込み制御部分を変更する。

【０１６０】

図１８は、割り込み禁止レベルを実現する第一のＯＳが管理しているデータ構造を示している。１８００は、割り込み禁止レベルテーブルである。それぞれの割り込みレベルは数値で表現され、それぞれの割り込み禁止レベルについて、何番の外部割り込みをマスクするかを示している。テーブル１８００のチェックのつけられているところは割り込みをマスクする設定することを示している。例えば、割り込み禁止テーブル１８００では、割り込み禁止レベル０では、どの割り込みもマスクされないことを示している。また、割り込み禁止レベル３では、割り込み番号３、ないし、５の割り込みを割り込み制御装置１１２によりマスクすることを示している。割り込み禁止レベル５では、すべての割り込みが割り込み制御装置１１２によりマスクされる。

【０１６１】

本発明では、第二のＯＳの初期化時に、この割り込み禁止レベルテーブル１８００を変更する（ステップ１４０９）。ステップ１４０９では、第二のＯＳが管理するデバイスが発生する割り込みについて、第一のＯＳがそれらの割り込みをマスクしないように割り込み禁止レベルテーブルを変更する。具体的には、割り込み識別テーブル１５２０のＯＳ欄１５２１を参照して、第二のＯＳが管理する割り込み番号について、割り込み禁止レベルテーブル１８００のチェックをクリアする。

【０１６２】

この例では、割り込み番号４と５が第二のＯＳが処理する割り込みとなっている。したがって、割り込み禁止レベルテーブル１８００のすべての割り込み禁止レベルの、割り込み番号４と５の欄（１８０１と１８０２のすべて）をクリアする。

【０１６３】

これにより、第一のＯＳが割り込み禁止レベルを変更しても、第二のＯＳが管理するデバイスの割り込みはマスクされなくなる。

【０１６４】

以上により、一台の計算機で２つのオペレーティングシステムを同時に動作させることが可能になる。

【０１６５】

本発明によれば、第一のＯＳに変更を加えて２つのオペレーティングシステムを同時に動作させる場合、変更箇所がオペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に２つのオペレーティングシステムを動作させることが可能になる。

【０１６６】

仮想計算機による方式では、物理メモリやＩ／Ｏチャネルを仮想化するために特権命令のエミュレーションが必要になるが、これをソフトウェアで実現するとオーバーヘッドが大きくなり問題である。仮想計算機方式では、このオーバーヘッド削減のために、特殊なハードウェアを持つことが多くなっている。しかし、本発明では、それぞれのデバイスについて、デバイスを管理するオペレーティングシステムを予め決定し、さらに、物理メモリについて利用できる範囲を初期化時に決定することにより、オペレーティングシステム同士が互いに干渉しないようにして、仮想計算機でのような複雑なソフトウェアによる制御

を廃止し、高速化のためのハードウェアも不要とした。

【０１６７】

本発明によれば、第一のＯＳの機能を補完するＯＳを容易に追加することが可能である。従来技術においても、第一のＯＳの構成要素として、例えばデバイスドライバとして新たな機能をカーネルに追加することは可能である。しかし、第一のＯＳの構成要素としてしまったのでは、その構成要素は第一のＯＳの管理下でしか動作できない問題がある。つまり、第一のＯＳが障害により停止してしまったとき、追加した機能モジュールも動作することはできないのである。

【０１６８】

本発明によれば、新たな機能を実現する構成要素を、第一のＯＳと独立して構成することができ、第一のＯＳが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。この実施形態については、後述する。信頼性を要求される機能モジュールを第二のＯＳとして組み込めば、第一のＯＳが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。このように本発明は、計算機システムの高信頼化を実現する手段となり得る発明である。

【０１６９】

また、この実施形態では、第二のＯＳが第一のＯＳに優先して処理を実行するとして説明した。第一のＯＳは、第二のＯＳがアイドル時にしか動作できないことや、第二のＯＳの割り込みはいつでもすぐに処理することにより、第二のＯＳを優先させている。これにより、第一のＯＳが実時間処理に適合していなくても、第二のＯＳとして実時間処理に向くＯＳを導入すれば、第一のＯＳの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築することが可能になる。例えば、第一のＯＳが優れたＧＵＩ（Ｇｒａｐｈｉｃａｌ　ＵｓｅｒＩｎｔｅｒｆａｃｅ）を持っているが実時間処理性能に欠ける場合、第一のＯＳよりも優先して動作する実時間処理向きオペレーティングシステムを第二のＯＳとして導入してやることで、ＧＵＩにも優れ、実時間処理にも優れた計算機システムの構築ができる。

【０１７０】

このように、本発明は、特別なハードウェアの支援なしで、第一のＯＳに欠けている機能を容易に導入する方法であり、さらに、その機能は第一のＯＳとは全く独立して動作させることを可能にする。

【０１７１】

次に、本発明の第二の実施形態について説明する。第二の実施形態は、これまで説明してきた実施の形態の拡張である。この実施の形態では、第一のＯＳが障害により停止しても動作しつづける第二のＯＳの導入が実現可能である。

【０１７２】

第一の実施形態に加えて、第一ＯＳ実行状態変数１５５０を共通領域に置く。この変数１５５０は、第一のＯＳが通常動作しているか、そうでないかを示す値を格納している。この変数１５５０は、第二のＯＳをロードする時の処理で、通常動作を示す値に初期化する。

【０１７３】

図１９は、本発明の第二の実施の形態の第一のＯＳの停止処理手順を示すフローチャートである。この処理手順は、第一のＯＳの停止処理を実行するモジュールを変更して実装する。

【０１７４】

まず、第一のＯＳの停止処理モジュールに制御がきたら、第一ＯＳ実行状態変数１５５０を第一のＯＳが停止していることを示す値に設定する（ステップ１９０１）。その後、第一のＯＳの停止処理を実行する（ステップ１９０２）。最後に、第一のＯＳへの割り込みをマスクして、第二のＯＳが処理するデバイスの割り込みを許可して（ステップ１９０３）、割り込みが発生するまで待つ（ステップ１９０４）。割り込みが発生すると、実行ＯＳの切り替えられ、第二のＯＳが処理を実行する。

【０１７５】
更に、実行ＯＳ切り替え手順を変更する。第一の実施の形態では、ステップ１６０１から始まる手順により実行ＯＳの切り替えを実行した。第二の実施の形態では、この手順で第二のＯＳのモジュールを実行した後、つまり、ステップ１６０５の後で、第一ＯＳ実行状態変数１５５０を検査する。ここで、第一ＯＳ実行状態変数１５５０が、第一のＯＳが停止していることを示す値になっているならば、ステップ１６０６以降の処理を実行せずに、割り込み待ちを実行する。
【０１７６】
以上のデータ構造と手順により、第一のＯＳが停止しても第二のＯＳの実行を継続することが可能になる。この実施形態では、第一のＯＳの停止処理モジュールを変更するとしたが、第一のＯＳがエラーにより停止したときの停止処理過程で実行されるモジュールを変更して、第一のＯＳの停止を検出して割り込み待ちをしても同様の効果を実現できる。
【０１７７】
本発明の第３の実施の形態について説明する。これまで説明してきた実施の形態では、カーネル本体を変更することにより２つのＯＳの同時実行等を実現してきた。第３の実施の形態では、カーネル本体を変更せずに、前記の実施の形態の機能を実現する。
【０１７８】
様々の種類のハードウェアをサポートするオペレーティングシステムでは、ハードウェア依存の処理がカーネル本体からは切り離されて、別のオブジェクトファイルとして構成されている場合がある。例えば、割り込み制御装置１１２が計算機により異なる場合や、バス１０９の構成が異なってＩ／Ｏアドレス空間が計算機により異なる場合である。
【０１７９】
図２０は、このようなオペレーティングシステム、つまり、割り込み制御装置やバスなどの基盤となるハードウェアの違いを吸収するためのコードやデータがカーネル本体とは分離されたオブジェクトファイルにある場合の、カーネル領域の様子を示した図である。
【０１８０】
カーネル領域２０００には、プロセッサ１０１のカーネルモードで実行されるモジュールや、オペレーティングシステムが管理するデータ構造がある。カーネル本体２００１は、メモリ管理、プロセススケジューリング、および、ファイルシステムなどの、ハードウェア非依存の処理を実施するコードやデータを持っている。カーネル本体２００１とハードウェア依存部２００２の間には、ハードウェア依存部２００２の提供しなければならないモジュールと、カーネル本体２００１が提供するモジュールに関する規約が定められている。ハードウェア依存部２００２をこの規約にあわせて構築すれば、様々な計算機上でこのオペレーティングシステムを動作させることが可能になる。
【０１８１】
この規約に従ったハードウェア依存の処理は、別オブジェクトファイルに分離され、カーネル本体とは切り離された領域２００２にマップされている。カーネル本体２００１とハードウェア依存部２００２は、第一の実施の形態の場合と同様の外部参照機構により互いの公開モジュールを呼び出すことができ、見かけ上は１つのカーネルとして機能する。
【０１８２】
このような場合は、カーネル本体のオブジェクトファイルを変更することなく、分離されたハードウェア依存の処理を実施するオブジェクトファイルの変更により第一の実施の形態、および、第二の実施の形態と同様の効果を得ることが可能である。
【０１８３】
具体的には、分離されたオブジェクトファイルの処理において、物理メモリの割り当てが可能であること、割り込みレベル管理処理を変更できること、および、Ｉ／Ｏ資源の予約ができることが必要である。さらに、このオブジェクトファイル中にステップ１７０１から始まる割り込みハンドラと割り込みテーブル１０７を配置し、プロセッサの割り込みテーブルレジスタ１０４に登録する。そして、この分離されたオブジェクトファイルを共通領域２０３として第二のＯＳからも参照できるようにする。以上により、本発明の第一の

実施の形態と同様の効果を得ることができる。

【０１８４】

さらに、ハードウェア依存オブジェクトファイルが、第一のＯＳが停止したときに実行されるモジュールを持つ規定になっていれば、そのモジュールを変更すれば第一のＯＳの停止を検出でき、本発明の第二の実施の形態と同様の効果を得ることができる。

【０１８５】

この実施の形態においては、カーネル本体を変更する必要がない。これにより、変更しなければならない部分が更に限定できる、カーネル本体を変更するよりも容易に実施可能となる。

【０１８６】

次に、本発明の第４の実施の形態について説明する。これまで説明してきた実施の形態では、共通領域２０３に配置していたのはサポートドライバや、ハードウェア依存オブジェクトファイルなどのオブジェクトファイルであった。しかし、本当に共通領域２０３に配置しなければならないモジュールとデータは、割り込みテーブル１０７、ステップ１７０１から始まる割り込みハンドラ、ステップ１６０１から始まるＯＳ切り替え手順、および、図１５に示したデータ構造分だけである。特に、第３の実施例でのようにハードウェア依存部の処理を実施するオブジェクトファイル全体を共通領域２０３として第二のＯＳからも参照できるようにしてしまうと、第二のＯＳが誤って第一のＯＳのデータ構造にアクセスしてしまう可能性が高くなり問題である。

【０１８７】

第４の実施の形態では、オブジェクトファイルの特定のセクションのみを共通領域２０３として第二のＯＳに見せる方法を提供する。この実施の形態では、オブジェクトファイルを生成するコンパイラが、命令コードとデータを配置するセクションをプログラム上で指定できる機能を持っている必要がある。

【０１８８】

通常のオブジェクトファイルは、セクションとして命令コードを含むテキストセクションと、データを含むデータセクションを持っている。これに加えて、コンパイラの機能により共通領域２０３のためのセクションを追加する。さらに、オブジェクトファイルのヘッダ部分に格納されているセクションデータ８０９を参照して共通領域セクションのアドレス範囲を決定し、その部分だけを第二のＯＳに見せるようにページテーブルを構築すればよい。

【０１８９】

ハードウェア依存処理をするモジュールを含むオブジェクトファイルを変更する場合を例として説明する。変更個所のうち初期化に関連する部分、例えば、物理メモリの割り当て、Ｉ／Ｏ資源の予約、割り込みレベル管理部分の変更は第二のＯＳに見せる必要はない。第二のＯＳからも参照できなければならないのは、割り込みテーブル１０７、ステップ１７０１から始まる割り込みハンドラ、ステップ１６０１から始まるＯＳ切り替え手順、および、図１５に示したデータ構造分だけである。これらを、共通領域セクションに配置するようにプログラムを記述し、コンパイラの機能により共通領域セクションを生成する。

【０１９０】

図２１は、生成されたオブジェクトファイルの構成を示している。２１００は、生成されたオブジェクトファイルを示す。オブジェクトファイル２１００のヘッダ部の２１０１、ないし、２１０４は、オブジェクトファイル２１００に含まれているセクションのデータを記述している。このうち２１０３と２１０４が、共通領域２０３用に新規に作成したセクションを表現するセクションデータである。対応するセクションは２１０７と２１０８である。セクションデータ２１０３と２１０４の内容にしたがってセクション２１０７と２１０８のアドレスを求め、それらの領域だけを第二のＯＳのカーネル領域にマップするように第二のＯＳのページテーブルを構成すれば、ハードウェア依存オブジェクトファイル２１００の他の部分を第二のＯＳから隠すことができる。

【０１９１】

第４の実施の形態によれば、これまで説明した実施の形態よりも、更にＯＳ間の独立性を高めることができ、ＯＳ間の干渉の少ない安全な計算機システムの構築が可能になる。

【０１９２】

次に、本発明の第５の実施の形態について説明する。第５の実施の形態では、マルチプロセッサ構成の計算機で第二のＯＳを導入が可能になる。

【０１９３】

図２２は、本発明の第５の実施の形態での計算機装置を示す図である。２２００は計算機装置である。計算機２２００は２つのプロセッサ２２０１と２２０２、および、主記憶装置２２０３を持っている。また、第一の実施の形態と同様に、計算機起動プログラムを格納している記憶装置２２０４を持っている。

【０１９４】

プロセッサ２２０１と２２０２について、プロセッサを起動したときと、初期化のための割り込みを受けた時とでは、制御を渡す物理アドレスが異なるものとする。

【０１９５】

記憶装置２２０４に格納されている初期化割り込み処理プログラムは、予め定めた物理アドレスに格納されている値を物理アドレスとして、そのアドレスに制御を渡す。

【０１９６】

また、バス２２０９を介して磁気ディスク装置２２０６、クロック割り込み生成装置２２０７、および、入出力装置２２０７等のデバイスが接続している。割り込みを発生するデバイスは、割り込み制御装置２２０５に接続し、更に、割り込みバス２２１１を介してプロセッサ２２０１と２２０２に接続している。各プロセッサは他のプロセッサに割り込みを送ることができるとする。

【０１９７】

割り込み制御装置２２０５について説明する。割り込み制御装置２２０５は、マルチプロセッサ構成のための機能を持っている。割り込み制御装置２２０５は、第一の実施の形態での割り込み制御装置１１２の割り込みマスク機能に加えて、それぞれのデバイスからの割り込みをどのプロセッサ、あるいは、プロセッサ群に通知するかを指定する機能を持っている。

【０１９８】

図２３は、割り込み制御装置２２０５の構成を示す図である。選択装置２３０１と割り込みマスクレジスタ２３０２の働きは、第１の実施の形態と同じである。それらに加えて、割り込み制御装置２２０５は、割り込み配送テーブル２３１０と、割り込み送信装置２３０５を持っている。

【０１９９】

割り込み配送テーブル２３１０は、割り込み制御装置２２０５に接続されたそれぞれのデバイスについて、どのプロセッサ、あるいは、プロセッサ群に割り込みを通知するかを示す値２３１１と、通知するときの割り込み番号２３１２を記録している。割り込み配送テーブル２３０２は、Ｉ／Ｏ命令により変更することができ、自由に設定可能である。

【０２００】

図２３の例では、割り込み０と１はＣＰＵ０に、割り込み２はＣＰＵ１に配送するように設定されている。

【０２０１】

割り込み送信装置２３０５は、選択装置２３０１からの信号を受けて、割り込み配送テーブル２３１０を参照して割り込み通知先と割り込み番号を決定する。そして、通知先と割り込み番号を表わす信号を割り込みバス２２１１へ送信する。

【０２０２】

計算機２２００は、起動するとプロセッサ２２０１だけが動作を開始するように構成されており、プロセッサ２２０１が記憶装置２２０４に格納されている起動プログラムを実行する。起動プログラムは、第一の実施の形態の場合と同様に磁気ディスク装置２２０６に格納されているカーネルローダを主記憶２２０３に読み込み実行する。カーネルローダは

、パラメータテーブル１１００を作成する。第５の実施の形態では、デバイスリストに計算機２２００が何個のプロセッサを持っているかを示すデータが加えられる。

【０２０３】

第一のＯＳのロード後、第一のＯＳの初期化処理を実行する。初期化の過程で、非ブートプロセッサ以外のプロセッサ用の初期化ルーチンのアドレスを予め定めた物理アドレスに格納し、プロセッサ２２０２に初期化割り込みを送る。プロセッサ２２０２は初期化割り込みを受けると、記憶装置２２０４に格納されているプログラムを実行し、非ブートプロセッサ初期化ルーチンに制御がわたる。非ブートプロセッサ初期化ルーチンは、ページテーブルレジスタや割り込みテーブルレジスタを設定して仮想アドレスモードに移行し、初期化処理を続ける。

【０２０４】

本発明の第５の実施の形態では、図１２のステップ１２０４の第二のＯＳ用のデバイスの予約のときに、プロセッサも第二のＯＳ専用であると予約する。ここでは、プロセッサ２２０２を予約するとして説明する。

【０２０５】

マルチプロセッサ構成の場合、ステップ１２０１から始まる第一のＯＳの初期化手順のシステムデバイスの初期化で、非ブートプロセッサに初期化割り込みを送る。この場合、プロセッサ２２０１からプロセッサ２２０２に初期化割り込みが送られることになる。本発明では、予約されているプロセッサについては初期化割り込みを送らないことにする。したかって、カーネルの初期化がされてもプロセッサ２２０２はまだ動作していない。

【０２０６】

また、ステップ１２０５のシステムデバイスの初期化では、割り込み制御装置２２０５の初期化も実施する。割り込み制御装置２２０５の初期化では、カーネル構成情報ファイル７００の第二のＯＳの構成データ７０４を参照して、第二のＯＳが管理するデバイスの割り込みがプロセッサ２２０２に送られるように割り込み配送テーブル２３１０を設定する。

【０２０７】

更に、図１４のステップ１４０１から始まる第二のＯＳの初期化手順において、初期化ルーチンを第二のＯＳの初期化ルーチンのアドレスに設定して、ステップ１４０７でプロセッサ２２０２に初期化割り込みを送る。これにより、プロセッサ２２０２上で、第二のＯＳが走行を開始する。

【０２０８】

第１、ないし、第４の実施の形態と異なり、第二のＯＳの管理するデバイスの割り込みは、すべて割り込み制御装置２２０５により、第二のＯＳが動作しているプロセッサ２２０２へ送られる。このため、実行ＯＳを切り替える必要はなくなる。第一のＯＳはプロセッサ２２０１で動作し、第二のＯＳはプロセッサ２２０２で動作することになる。したがって、ステップ１７０１から始まる割り込み処理も不要になる。

【０２０９】

第二のＯＳは、独自の割り込みテーブルをプロセッサ２２０２の割り込みテーブルレジスタに設定し、独自の割り込みハンドラを持てる。第一のＯＳの割り込みテーブルを変更する必要はない。但し、第一のＯＳが割り込み制御装置２２０５の割り込みマスクレジスタ２３０２を変更する場合は、第二のＯＳのデバイスからの割り込みをマスクしてしまわないように変更を加える必要がある。

【０２１０】

第５の実施の形態においては、第１ないし４の実施の形態よりも、性能の良い計算機システムの構築か可能である。第１ないし４の実施の形態では、第一のＯＳは第二のＯＳがアイドルしている間のみ動作可能であったが、第５の実施の形態においては、第一のＯＳは、プロセッサは奪われてはいるが常に動作することが可能であり、同時に第二のＯＳも動作可能である。

【０２１１】

更に、第５の実施の形態によれば、第一のＯＳと第二のＯＳとで共有しなければならない領域を小さくできる。第１ないし４の実施の形態では、共通領域２０３に、割り込みテーブル、割り込みハンドラや割り込み処理に付随するデータ構造、および、ＯＳ切り替えコードを置かなければならなかった。第５の実施の形態では、これらはすべて必要なくなり、互いのＯＳが相手のＯＳを誤って破壊してしまう可能性を低くできる。

【０２１２】

次に、本発明の第６の実施の形態について説明する。これまで説明した実施形態が２つのＯＳを同時実行する方式であるのに対し、第６の実施形態は２つ以上の複数のＯＳを同時実行する方式である。

【０２１３】

第一の実施形態では、第二のＯＳが第一のＯＳよりも優先して実行するように制御するが、ここで説明する方式では、複数のＯＳ間に実行優先度を設定可能である。例えば、割り込みについては、実行中のＯＳの優先度よりも低い優先度のＯＳが管理する割り込みの処理は延期される。実行中のＯＳの優先度よりも高い優先度のＯＳが管理する割り込みが発生した場合は、即座に実行ＯＳを切替え割り込み処理を開始する。

【０２１４】

また、実行中のＯＳが自分よりの優先度の高いＯＳのモジュールを呼び出す場合は即座に実行ＯＳを切替えてモジュール呼び出しを実施する。その逆の場合、つまり、優先度の低いＯＳの側での処理が必要になる場合は、そのＯＳが実行権を得るまで要求された処理を延期するように制御する。

【０２１５】

図２４は、本発明の第６の実施形態の計算機構成を示した図である。計算機構成はこれまでの実施例と同じであるが、主記憶装置１０２に複数のＯＳがロードされている様子を示している。各オペレーティングシステムは、ステップ１４０１に示した第二のＯＳをロードする手順と同じ手順でロードできる。

【０２１６】

図２５は、複数のＯＳの関係を概念的に示した図である。第一の実施の形態では２つのＯＳであったのに対し、ここでは第一のＯＳ以外に複数のＯＳが１つのプロセッサ上で動作していることを示している。

【０２１７】

第一のＯＳの一部である共通領域が他の第２、第３、および、第ＮのＯＳの論理空間２０２、２５０３、および、２５０４にマップされ全ＯＳから共通に利用できることを示している。各ＯＳの論理空間への共通領域のマッピングは、ステップ１４０１に示した手順により実施する。

【０２１８】

更に、各ＯＳはそれぞれが管理する外部機器を持っていることを示している。第二のＯＳは機器１１６と１１７、第三のＯＳは機器２５０５と２５０６、第ＮのＯＳは機器２５０７を管理することを示している。これらの機器を制御するためのＩ／Ｏアドレス範囲、および、割り込み番号は、図７に示すカーネル構成情報ファイル７００に格納しておけば良い。図７では第二のＯＳ用の構成情報のみを格納しているように記述したが、他に第三、第四のＯＳの構成情報を格納しておく。ステップ１２０１からの初期化手順では、第二のＯＳだけでなく第一のＯＳ以外の全てのＯＳの資源を予約して、第一のＯＳが第一のＯＳ以外のＯＳが管理するデバイスへアクセスすることを禁止する。

【０２１９】

ステップ１２０１から始まる手順では、ステップ１２０２で第二のＯＳ用の主記憶を確保している。これを、第一のＯＳ以外の複数のＯＳ用の主記憶を確保する処理とする。また、ステップ１２０４は第二のＯＳが管理するデバイスを予約する処理であるが、これを第一のＯＳ以外のＯＳが管理するデバイス資源を予約する処理とする。

【０２２０】

本実施形態における第一のＯＳの初期化手順を図２６に示す。ステップ２６０２と２６０

４がステップ１２０２と１２０４に対応している。ステップ２６０４は、カーネル構成情報ファイル７００を参照して第一ＯＳ以外のＯＳのデバイス資源を予約する。

【０２２１】

この時、同時に割り込み管理テーブル１５２０の処理ＯＳ１５２１も設定する。また、各ＯＳの構成情報には、デバイス資源の他にＯＳの優先度も記述する。

【０２２２】

図２７は、共通領域２０３に配置するデータ構造を示す図である。図１５に示したデータ構造と比較して、割り込み識別テーブル１５２０、ＯＳ識別変数１５３０、および、遅延割り込み状態変数１５４０は同一のデータ構造である。ＯＳコンテクストテーブル２７１０は、１５１０を拡張したデータ構造となっている。

【０２２３】

テーブル２７１０は、各ＯＳの実行を切替える時に必要になるデータを保存している。ページテーブル設定値２７０１とスタックポインタ設定２７０２は、あるＯＳを実行中に他のＯＳのモジュールを呼び出す時に設定するページテーブルとスタックポインタのアドレスを示している。また、ページテーブル保存値２７０３とスタックポインタ保存値２７０４は、優先度の高いＯＳへの切替を実施した時の優先度が低い方のＯＳのページテーブル値、および、スタックポインタ値を保存している。

【０２２４】

実行状態２７０５は、それぞれのＯＳについて稼働中であるか、および、処理待ち中であるかを示す値を格納する。ここで稼働中であるかとは、ＯＳが起動されているかを示す。ある瞬間に実行中であることを示しているわけではない。各ＯＳの起動処理は実行状態２７０５を設定する。

【０２２５】

また処理待ち中であるとは、そのＯＳがアイドル状態でないことを示す。つまり、優先度の高いＯＳが実行中のため走行待ち状態にあることを示す。処理待ち中については、具体的にどの処理が待ちになっているかを記述してもよい。

【０２２６】

優先度２７０６は、各ＯＳの実行優先度を格納する。優先度は、第一ＯＳの初期化処理手順のステップ２６０５で、構成ファイル７００より読み出し設定する。

【０２２７】

実行ＯＳの切替え手順について、第二のＯＳが実行中であるとして説明する。また、ＯＳの実行優先度が第一より第二、第二より第三の方が優先度が高く設定されているとする。第二のＯＳの処理により第三のＯＳ上のプロセスが走行可能になったとする。この場合、第三のＯＳ内部のプロセス起動モジュールを実行して第三のＯＳ上のプロセスをスケジュールする。ここで、第三のＯＳは第二のＯＳよりも優先度が高いので、実行ＯＳを即座に切替えて第三のＯＳのモジュールが実行される。切替え処理では現在のページテーブルアドレスとスタックポインタ値を、第二のＯＳのページテーブル保存値２７０３とスタックポインタ保存２７０４値に格納し、第三のＯＳのページテーブル設定値２７０１とスタックポインタ設定値２７０２をページテーブルレジスタとスタックポインタに設定して実行ＯＳを切替える。

【０２２８】

第二のＯＳが第一のＯＳのモジュールを呼び出す場合、第一のＯＳの優先度は第二のＯＳの優先度よりも低いため、呼び出しは延期する。この場合、呼び出し要因を実行状態２７０５記録しておき、第一のＯＳが実行権を得た時に、すなわち、第二、および、第三のＯＳが処理を終了した時に、モジュール呼び出しを実施する。

【０２２９】

図２８は、実行ＯＳの切替え手順を示すフローチャートである。ステップ２８０１からの処理の大部分は図１６に示したステップ１６０１からの処理と同じである。この手順は、ＯＳの優先度については考慮していないが、この手順を呼び出す前に呼出先ＯＳの優先度と実行中ＯＳの優先度を比較して、実際に呼び出して良い場合はステップ２８０１からの

処理を実施すれば良い。

【０２３０】

ステップ２８０１から始まる処理は、切替先ＯＳと呼び出しモジュールのアドレスを引数として受ける。ステップ２８０１では、現在のページテーブルアドレスとスタックポインタを、ＯＳコンテクストテーブル２７１０の現在実行中のＯＳのページテーブル保存値２７０３と、スタックポインタ保存値２７０４に保存する。現在実行中のＯＳは、ＯＳ識別変数１５３０により判定できる。

【０２３１】

次のステップ２８０２では、ＯＳコンテクストテーブル２７１０より切替先ＯＳのページテーブル設定値２７０１とスタックポインタ設定値２７０２を取得し、ページテーブルとスタックの切替え処理を実施する。

【０２３２】

ステップ２８０３では、遅延割り込み状態をクリアする。優先度が実行中のＯＳ以上で、切替先未満のＯＳが管理するデバイスの遅延割り込み状態１５４０をクリアする。

【０２３３】

例えば、第一のＯＳから第三のＯＳのモジュールを呼び出すとする。この場合、ステップ２８０３では第一のＯＳと第二のＯＳが管理するデバイスの遅延割り込み状態１５４０をクリアする。

【０２３４】

続くステップ２８０４では、ＯＳコンテクストテーブル２７１０の現在実行中のＯＳの実行状態２７０５を処理待ち中に設定し、ステップ２８０５でＯＳ識別変数を切替先ＯＳに設定し、ステップ２８０６で引数として渡されたモジュールを呼び出す。

【０２３５】

切替先のＯＳは実施すべき処理がなくなった時点で、ステップ２８０７へ制御を戻す。ステップ２８０７では、ＯＳコンテクストテーブルの各ＯＳの実行状態２７０５と優先度２７０６を参照して、最も優先度が高い処理待ち中のＯＳを見つける。

【０２３６】

ステップ２８０８は、ステップ２８０７で選択したＯＳのコンテクストを、ＯＳコンテクストテーブル２７１０のページテーブル保存値２７０３とスタックポインタ保存値２７０４より回復する。

【０２３７】

次のステップ２８０９で、ＯＳ識別変数１５３０を選択したＯＳに設定する。

【０２３８】

ＯＳコンテクストテーブル２７００の実行状態２７０５に延期されている処理があることが記録されている場合は、その延期されている処理を実行する（ステップ２８１０）。

【０２３９】

続く処理では、遅延された割り込みを処理する。ステップ２８１２は、遅延割り込み状態１５４０を参照して、ステップ２８０７で選択されたＯＳが処理すべき割り込みが発生していないかを検査する。選択されたＯＳがどの割り込み番号の割り込みを管理しているかは、割り込み識別テーブル１５２０の処理ＯＳ１５２１を参照すれば分かる。

【０２４０】

ステップ２８１２の検査の結果、処理待ちの割り込みが発生していると判定した場合はステップ２８１３へ進む。ステップ２８１３では、処理しなければならない割り込みが発生していることを選択されたＯＳが認識できるように選択されたＯＳのデータ構造を設定する。続くステップ２８１４は、割り込み識別テーブル１５２０を参照して、処理する割り込みの割り込みハンドラ１５２２を呼び出す。ハンドラの処理終了後、回復したコンテクストにしたがって、選択されたＯＳの実行を再開する。

【０２４１】

処理待ち割り込みがなければ、そのまま回復したコンテクストにしたがって、選択されたＯＳの実行を再開する。

【０２４２】
割り込み処理について説明する。図２９は、割り込み処理手順を示すフローチャートである。図２９に示す手順を実行するルーチンはプロセッサ１０３の割り込みハンドラとして割り込みテーブル４００に登録される。また、全てのＯＳから参照可能な共通領域に配置される。
【０２４３】
処理手順について説明する。まず、ステップ２９０１では、割り込み要因より割り込みを処理するＯＳを決定する。処理ＯＳは、割り込み識別テーブル１５２０の処理ＯＳ１５２１に記録されており、これを参照して決定する。
【０２４４】
割り込み発生時のＯＳと割り込み処理ＯＳが同一ならば（ステップ２９０２）、割り込み識別テーブル１５２０に登録されている割り込みハンドラ１５２２を実行する（ステップ２９０３）。ハンドラ処理終了時に割り込まれた処理を再開する。
【０２４５】
割り込み発生時のＯＳと割り込み処理ＯＳが異なる場合は、２つのＯＳの優先度を比較する（ステップ２９０４）。実行中のＯＳの優先度の方が高い場合、ステップ２９０５へ進み、遅延割り込み状態変数１５４０を設定して、割り込まれた処理を再開する。
【０２４６】
割り込み処理ＯＳの優先度の方が実行中のＯＳよりも高い場合は、ステップ２９０６へ進み、割り込み処理を開始する。
【０２４７】
ステップ２９０６では、割り込み識別テーブル１５２０より割り込みハンドラアドレス１５２２を取得する。続いて、割り込み発生時のコンテクストを保存し（ステップ２９０７）、割り込みハンドラを呼び出す（ステップ２９０８）。割り込みハンドラの呼び出しは、図２８のステップ２８０１から始まる手順により実施する。
【０２４８】
その後、割り込まれたＯＳに制御が戻った時に、割り込み発生時のコンテクストを回復し（ステップ２９１０）、割り込まれた処理を再開する。
【０２４９】
次に、共通領域２０３に配置する処理モジュールについて述べる。共通領域２０３には、ステップ２８０１からのＯＳ切替えモジュールと２９０１から始まる割り込みハンドラを配置する。この他の全てのＯＳから呼び出されるモジュールも共通領域２０３に配置する。例えば、各ＯＳの割り込みハンドラを割り込み識別テーブル１５２０に登録するモジュールや、ＯＳコンテクストテーブル２７１０の実行状態２７０５を設定するモジュールである。
【０２５０】
各ＯＳは、それぞれの初期化処理で共通領域２０３の登録モジュールを呼び出して割り込みハンドラを登録することにより、外部割り込みの処理が可能になる。また、何らかの処理により実行中のＯＳの優先度よりも低い優先度の他のＯＳの処理が可能になる場合、実行状態２７０５に処理待ち中であること、あるいは、処理待ちの要因を示す値を設定しなければならない。この場合も、共通領域２０３内の実行状態設定モジュールを呼び出すことにより設定する。
【０２５１】
本実施形態によれば、第一のＯＳの他に複数のＯＳを１つのプロセッサで実行可能となり、特別な機能に特化したＯＳを組み合わせることが可能となる。例えば、リアルタイム性に優れたＯＳと計算機の信頼性を補完するＯＳとを組み合わせて、第一のＯＳの機能を補うことができる。各ＯＳは独立しているのでさまざまのＯＳを組み合わせることができ、用途に応じた第一のＯＳの機能拡張が可能になる。更に、これまで説明した実施形態により得られる効果も損なわれることはない。
【０２５２】

更に、ＯＳ間に優先度を設定できるので、リアルタイムＯＳの優先度を最も高く設定するといった設定も可能である。これにより、各ＯＳの優れている機能を有効に利用することが可能になる。

【０２５３】

本発明の第７の実施の形態について説明する。本実施形態は、第６の実施の形態の拡張である。第７の実施の形態では、複数動作している内のあるＯＳが障害停止した場合に、他の残ったＯＳが実行継続可能な制御方法を示す。

【０２５４】

各ＯＳの障害処理モジュールは、ＯＳの実行を停止する時に、ＯＳコンテクストテーブル２７１０の実行状態２７０５を、停止中であると設定しなければならない。これは、前記実施形態の、共通領域２０３に配置した実行状態設定モジュールの呼び出しにより実施する。

【０２５５】

また、各ＯＳは、他のＯＳが実行を停止した時に呼び出されるＯＳ停止通知ハンドラを持ち、割り込みハンドラと同様に共通領域２０３内のデータ構造に登録する。

【０２５６】

次に、実行状態設定モジュールの処理について説明する。図３０は、本モジュールの処理を示したフローチャートである。第三のＯＳが停止したとして説明する。

【０２５７】

ステップ３００１では、ＯＳコンテクストテーブル２７１０の実行状態２７０５を設定する。

【０２５８】

ステップ３００２では、設定された実行状態を検査する。もし、ＯＳ実行停止に設定されたらならば、ステップ３００３へ進む。そうでなければモジュールの処理は終了する。

【０２５９】

ステップ３００３では、停止したＯＳよりも優先度が高い稼働中のＯＳのＯＳ停止通知ハンドラを呼び出す。呼び出しは、ステップ２８０１からのＯＳ切替えにより呼び出す。優先度が低いＯＳについては、実行状態２７０６にＯＳ停止ハンドラ実行待ちであることを記録する。また、各ＯＳのＯＳ停止通知ハンドラは、割り込みハンドラと同様に、各ＯＳの初期化時に共通領域内に配置されたデータ領域に登録されるものとする。

【０２６０】

実行ＯＳ切替え処理では、切替先のＯＳが実行を停止していないかを検査する処理が必要である。図２８のフローチャートにしたがって説明する。

【０２６１】

まず、切替え処理を始める前に、切替先が稼働中かを検査しなければならない。ステップ２８０１の前に、ＯＳコンテクストテーブル２７１０の実行状態２７０５を検査して、稼働中でなければ切替え処理を終了させる。

【０２６２】

更に、ステップ２８０７での次の実行ＯＳの選択では、処理待ち中のＯＳの検索対象から稼働していないＯＳを除外するようにする。

【０２６３】

選択したＯＳの実行を再開する前に、選択したＯＳの実行状態２７０５を検査し、ＯＳ停止通知ハンドラの実行待ちが記録されているならば、ＯＳ停止通知ハンドラを実行する。

【０２６４】

次に、割り込み処理について説明する。図２９のフローチャートにしたがって説明する。ステップ２９０１の割り込み処理ＯＳ決定の後、処理ＯＳが稼働中か検査する。ここで稼働中でないと判定した場合、割り込みを解除して割り込まれた処理を再開させるようにする。

【０２６５】

以上の処理により、第一のＯＳ以外に複数のＯＳが１つのプロセッサ上で同時に動作して

いる時にいくつかのＯＳが障害により停止しても、他の残りのＯＳは動作を継続することが可能になる。

【０２６６】

また、各ＯＳの障害停止が他のＯＳに通知されるので、ＯＳ間で連携して処理を実施している場合でも、その通知をもとに各ＯＳで障害処理を実施することが可能であり、計算機全体としての信頼性を高めることができる。

【０２６７】

これまで述べた実施の形態では、それぞれのＯＳは互いに異なるＯＳであっても、同一のＯＳが含まれていても良い。

【０２６８】

【発明の効果】

本発明は、特別なハードウェアの支援なしで、第一のＯＳに欠けている機能を容易に導入する方法であり、さらに、その機能は第一のＯＳとは全く独立して動作させることを可能にする。

【０２６９】

本発明によれば、第一のＯＳに対する変更点は、オペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に２つのオペレーティングシステムを同時に動作させることが可能になる。

【０２７０】

仮想計算機による方式で発生するオーバーヘッドについて、本発明では、それぞれのデバイスについて、デバイスを管理するオペレーティングシステムを予め決定し、さらに、物理メモリについて利用できる範囲を初期化時に決定することにより、オペレーティングシステム同士が互いに干渉しないようにして、仮想計算機でのような複雑なソフトウェアによる制御を廃止し、命令エミュレーションによるオーバーヘッドも削減し、高速化のためのハードウェアも不要とした。

【０２７１】

従来技術においても、第一のＯＳの構成要素として新たな機能をカーネルに追加することは可能であるが、第一のＯＳが停止してしまったとき、追加した機能モジュールも動作することはできない問題があった。本発明によれば、新たな機能を実現する構成要素を、第一のＯＳと独立して構成しておけば、第一のＯＳが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。信頼性が要求される機能モジュールを第二のＯＳとして組み込めば、第一のＯＳが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。

【０２７２】

また、第一のＯＳは、第二のＯＳがアイドル時にしか動作できないことや、第二のＯＳの割り込みはいつでもすぐに処理することにより、第二のＯＳを優先させることができる。これにより、第一のＯＳが実時間処理に適合していなくても、第二のＯＳとして実時間処理に向くＯＳを導入すれば、第一のＯＳの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築することが可能になる。

【０２７３】

また、第一のＯＳのカーネルが、カーネル本体とハードウェア依存の処理を実行するオブジェクトファイルとに分離されている場合、カーネル本体を変更しなくとも、後者のオブジェクトファイルを変更することで２つのオペレーティングシステムを同時に動作させることが可能になる。これにより、更に変更しなければならない部分が限定されるため、カーネル本体を変更するよりも容易に実施可能となる。

【０２７４】

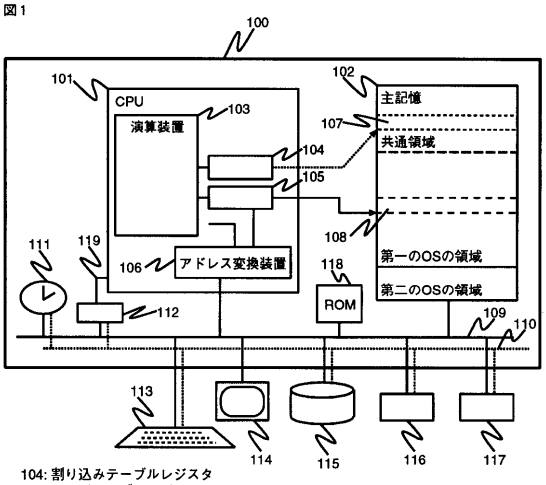
また、プログラムの記述によりオブジェクトファイル中に自由にセクションを設けることができるならば、それを利用して第一のＯＳと第二のＯＳとで共通に参照する必要のある領域を特別なセクションに閉じ込めることで、ＯＳ間の独立性を高めることができ、ＯＳ間の干渉の少ない安全な計算機システムの構築が可能になる。

【０２７５】
本発明によれば、第一のＯＳに対する変更点は、オペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に複数のオペレーティングシステムを同時に動作させることが可能になる。
【０２７６】
複数のオペレーティングシステムを動作実行させた場合でも、上に示した効果を得ることができる。
【０２７７】
従来技術においても、第一のＯＳの構成要素として新たな機能をカーネルに追加することは可能であるが、第一のＯＳが停止してしまったとき、追加した機能モジュールも動作することはできない問題があった。本発明によれば、新たな機能を実現する構成要素を、第一のＯＳと独立して構成しておけば、第一のＯＳが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。信頼性が要求される機能モジュールを第二、第三のＯＳとして組み込めば、第一のＯＳが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。
【０２７８】
また、第一のＯＳの他に複数のＯＳを１つのプロセッサで実行可能なことにより、特別な機能に特化したＯＳを組み合わせることが可能となる。例えば、リアルタイム性に優れたＯＳと計算機の信頼性を補完するＯＳとを組み合わせて、第一のＯＳの機能を補うことができる。各ＯＳは独立しているのでさまざまのＯＳを組み合わせることができ、他の効果を損なうことなく用途に応じた第一のＯＳの機能拡張が可能になる。
【０２７９】
また、複数のＯＳ間に実行優先度を設定できることにより、第一のＯＳが実時間処理に適合していなくても、実時間処理に向く第二のＯＳを最高優先度に設定すれば、第一、および、第二のＯＳ以外のＯＳの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築できすることが可能となる。
【０２８０】
また、第一のＯＳのカーネルが、カーネル本体とハードウェア依存の処理を実行するオブジェクトファイルとに分離されている場合、カーネル本体を変更しなくとも、後者のオブジェクトファイルを変更することで複数のオペレーティングシステムを同時に動作させることが可能になる。また、プログラムの記述によりオブジェクトファイル中に自由にセクションを設けることができるならば、それを利用して全てのＯＳで共通に参照する必要のある領域を特別なセクションに閉じ込めることで、ＯＳ間の独立性を高めることができ、ＯＳ間の干渉の少ない安全な計算機システムの構築が可能になる。
【図面の簡単な説明】
【図１】本発明の実施の形態の、計算機構成を示す図である。
【図２】本発明の実施の形態の、計算機構成を示す図である。
【図３】本発明の実施の形態の、ページテーブルの構成を示す図である。
【図４】本発明の実施の形態の、割り込みテーブルの構成を示す図である。
【図５】本発明の実施の形態の、割り込み制御装置の構成を示す図である。
【図６】本発明の実施の形態の、計算機のブート手順を示すフローチャートである。
【図７】本発明の実施の形態の、第一のＯＳのカーネル構成情報ファイルの構成を示す図である。
【図８】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。
【図９】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。
【図１０】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。
【図１１】本発明の実施の形態の、カーネル起動パラメータテーブルのデータ構造を示す図である。
【図１２】本発明の実施の形態の、第一のＯＳの初期化手順を示すフローチャートである。

【図１３】本発明の実施の形態の、第一のＯＳのデバイス管理テーブルのデータ構造を示す図である。
【図１４】本発明の実施の形態の、第二のＯＳの起動手順を示すフローチャートである。
【図１５】本発明の実施の形態の、第一のＯＳと第二のＯＳが共有するデータ構造を示す図である。
【図１６】本発明の実施の形態の、実行ＯＳの切り替え手順を示すフローチャートである。
【図１７】本発明の実施の形態の、割り込み処理手順を示すフローチャートである。
【図１８】本発明の実施の形態の、第一のＯＳの割り込みマスク処理のためのデータ構造を示す図である。
【図１９】本発明の第２の実施の形態の、第一のＯＳの障害停止処理を示すフローチャートである。
【図２０】本発明の第３の実施の形態の、第一のＯＳと第二のＯＳのカーネル領域の構成を示す図である。
【図２１】本発明の第４の実施の形態の、オブジェクトファイルの構成を示す図である。
【図２２】本発明の第５の実施の形態の、計算機システムの構成を示す図である。
【図２３】本発明の第５の実施の形態の、割り込み制御装置の構成を示す図である。
【図２４】本発明の第６の実施の形態の、計算機構成を示す図である。
【図２５】本発明の第６の実施の形態の、計算機構成を示す図である。
【図２６】本発明の第６の実施の形態の、第一のＯＳの初期化手順を示すフローチャートである。
【図２７】本発明の第６の実施の形態の、全てのＯＳが共有するデータ構造を示す図である。
【図２８】本発明の第６の実施の形態の、実行ＯＳの切り替え手順を示すフローチャートである。
【図２９】本発明の第６の実施の形態の、割り込み処理手順を示すフローチャートである。
【図３０】本発明の第６の実施の形態の、ＯＳ実行状態設定手順を示すフローチャートである。
【符合の説明】
１００は計算機、１０１はプロセッサ、１０２は主記憶装置、１０３は演算装置、１０４は割り込みレジスタ、１０５はページテーブルレジスタ、１０６はアドレス変換装置、１０７は割り込みテーブル、１０８はページテーブル、１０９はバス、１１０は割込み信号線、１１１はクロック生成装置、１１２は割り込み制御装置、１１３ないし１１７は外部入出力装置、１１８は割り込みバスである。

【図１】

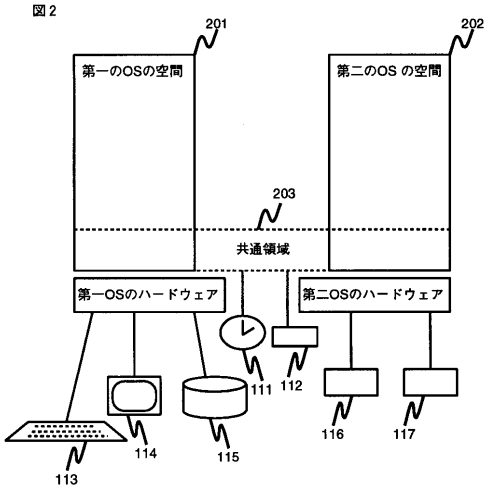
図1



104: 割り込みテーブルレジスタ
105: ページテーブルレジスタ
112: 割り込みコントローラ

【図２】

図2



【図３】

図3

300

仮想ページ番号	有効	物理ページ番号
0	✓	0
1	✓	56
2	✓	23
3		
4		
5	✓	32
		⋮

300: ページテーブル

【図４】

図4

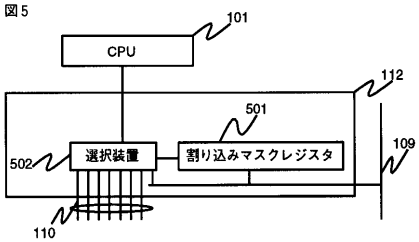
400

割込番号	割り込みハンドラ開始アドレス
0	800h
1	830h
2	500h
3	
4	
5	

400: 割り込みテーブル

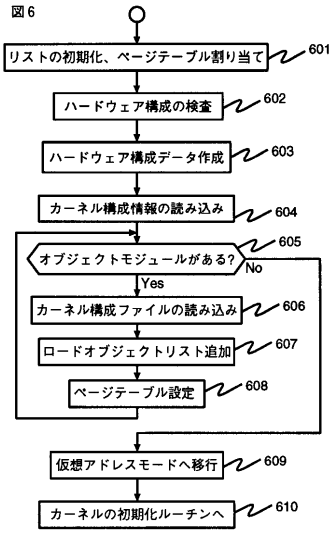
【図５】

図5



【図６】

図6



【図７】

図7

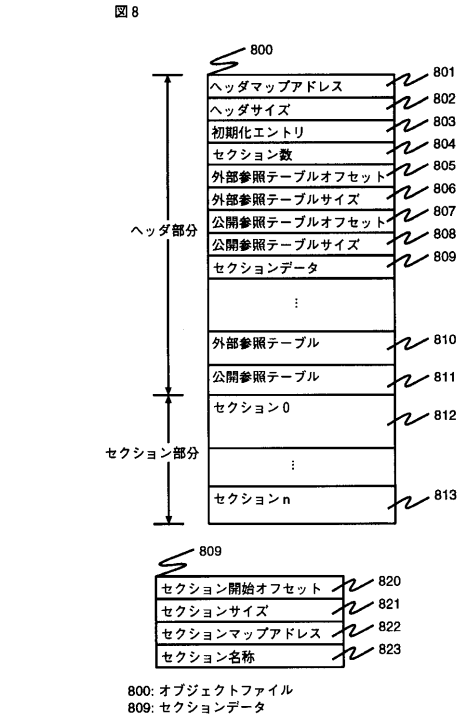
700

名前	内容
オブジェクトファイル	kernel, driver1, driver2
driver2	driver2固有データ
secondary OS	第二OS固有データ

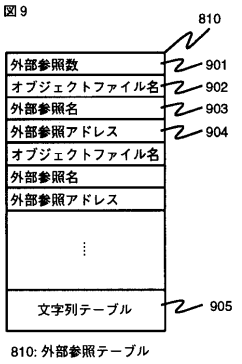
701 702 703 704

700: カーネル構成情報ファイル

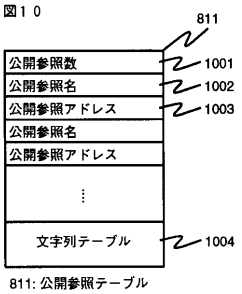
【図８】



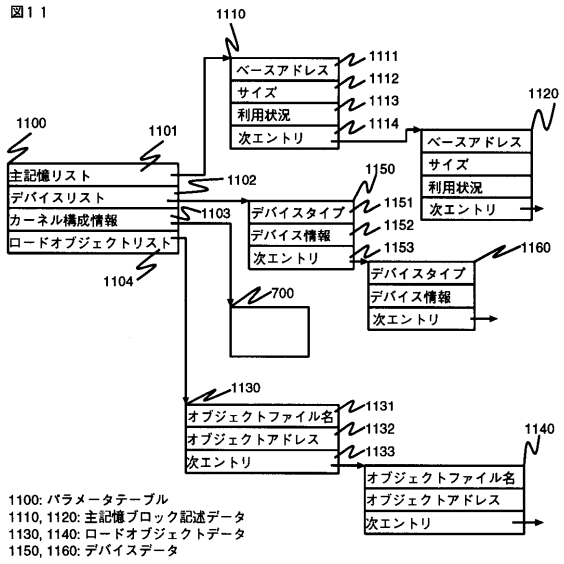
【図９】



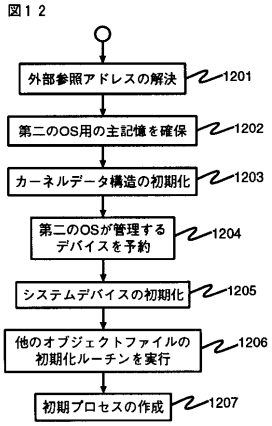
【図１０】



【図１１】

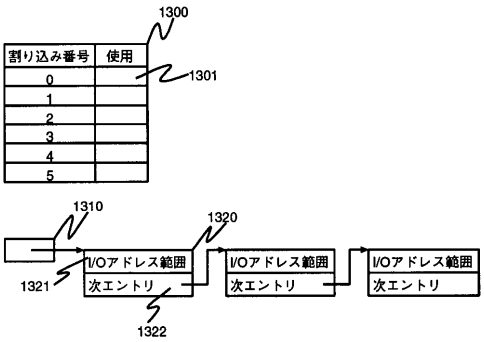


【図１２】



【図１３】

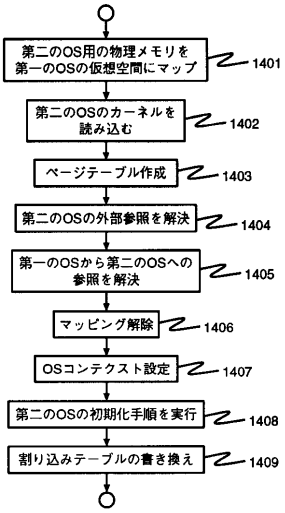
図１３



1300: 割り込みベクタ管理テーブル
1310: I/Oアドレス管理リスト

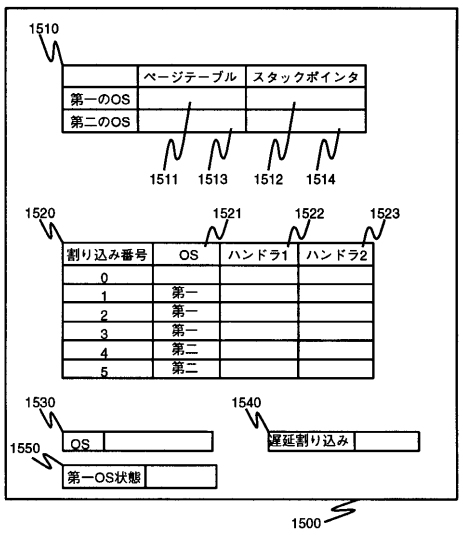
【図１４】

図１４



【図１５】

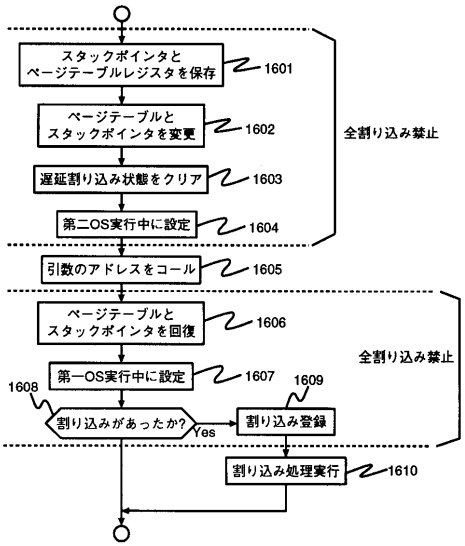
図１５



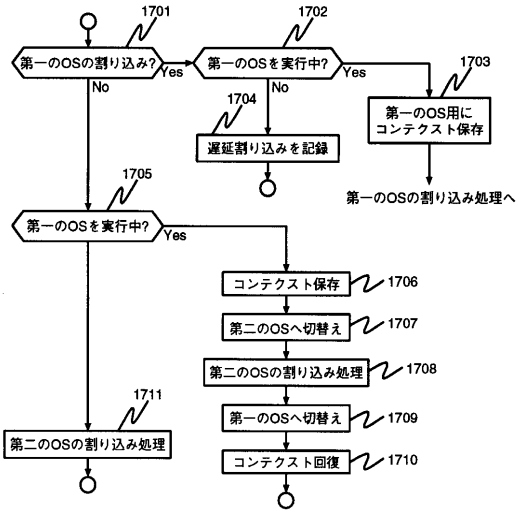
1500: 共通領域
1510: OSコンテクストテーブル
1520: 割り込み識別テーブル
1530: OS識別変数
1540: 遅延割り込み状態変数
1550: 第一OS状態変数

【図１６】

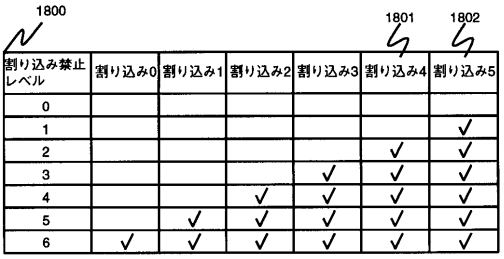
図１６



【図１７】
図１７



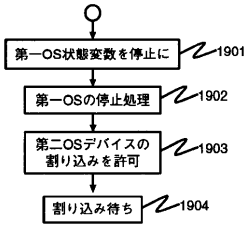
【図１８】
図１８



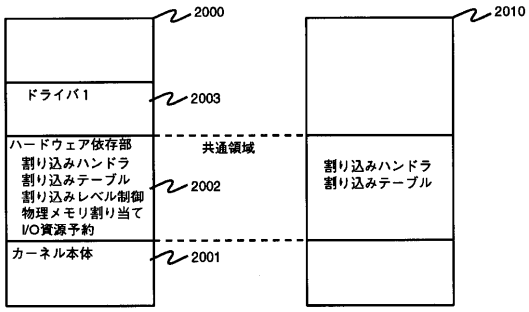
割り込み禁止レベル	割り込み0	割り込み1	割り込み2	割り込み3	割り込み4	割り込み5
0						
1						✓
2					✓	✓
3				✓	✓	✓
4			✓	✓	✓	✓
5		✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓

1800: 割り込み禁止レベルテーブル

【図１９】
図１９

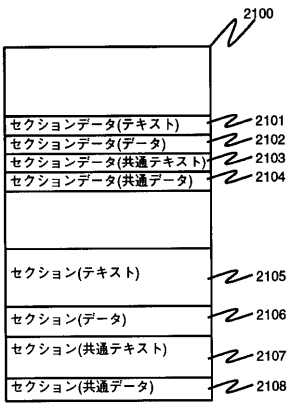


【図２０】
図２０



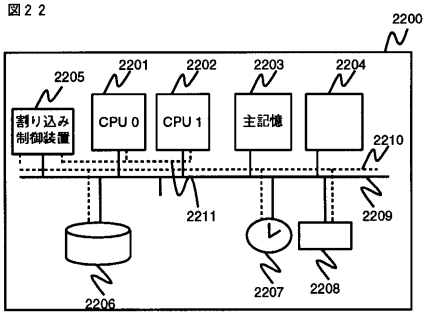
2000: 第一のOSのカーネル領域
2010: 第二のOSのカーネル領域

【図２１】
図２１

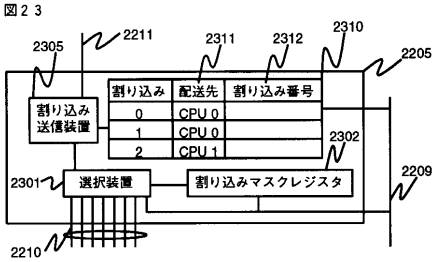


2100: ハードウェア依存オブジェクトファイル

【図２２】
図２２



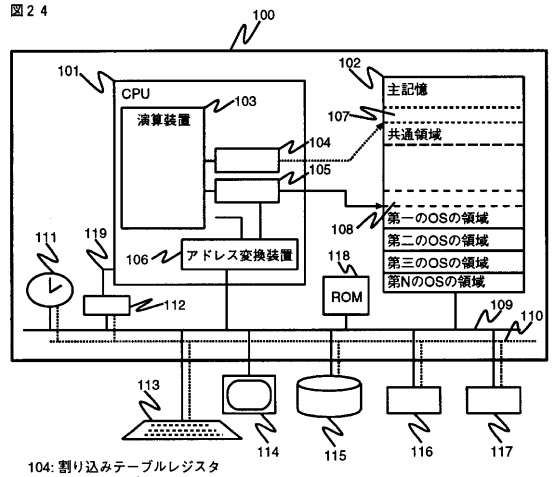
【図２３】
図２３



2205: 割り込み制御装置
2310: 割り込み配送テーブル

【図２４】

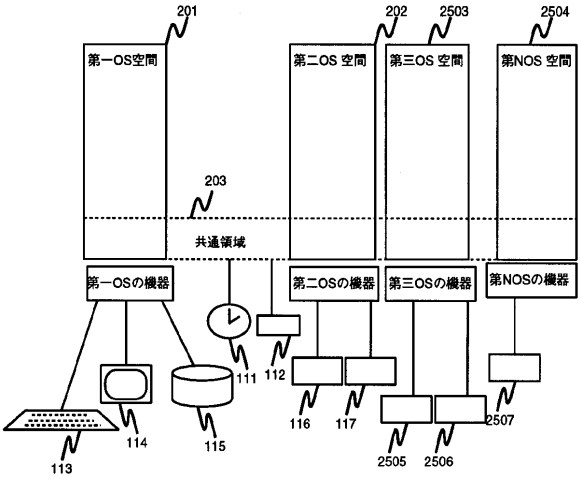
図２４



104: 割り込みテーブルレジスタ
105: ページテーブルレジスタ
112: 割り込みコントローラ

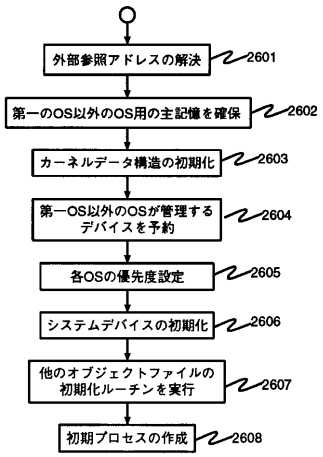
【図２５】

図２５



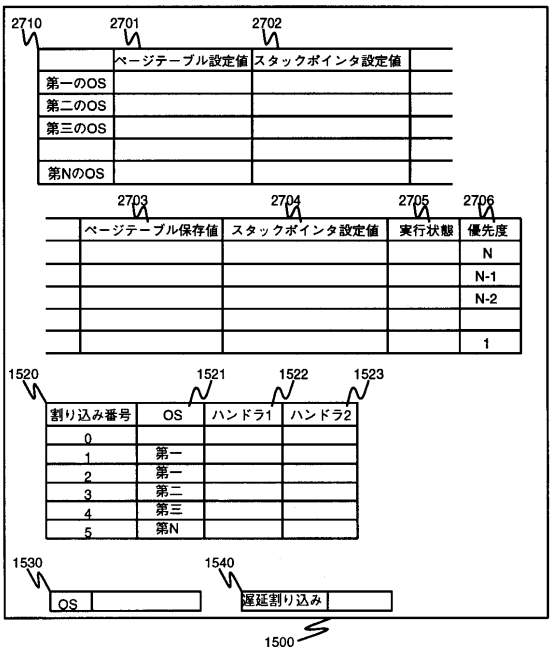
【図２６】

図２６



【図２７】

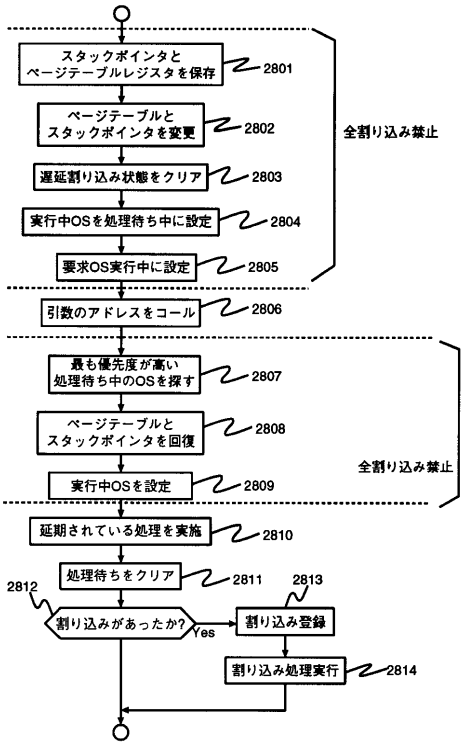
図２７



1500: 共通領域
2710: OSコンテクストテーブル
1520: 割り込み識別テーブル
1530: OS識別変数
1540: 遅延割り込み状態変数

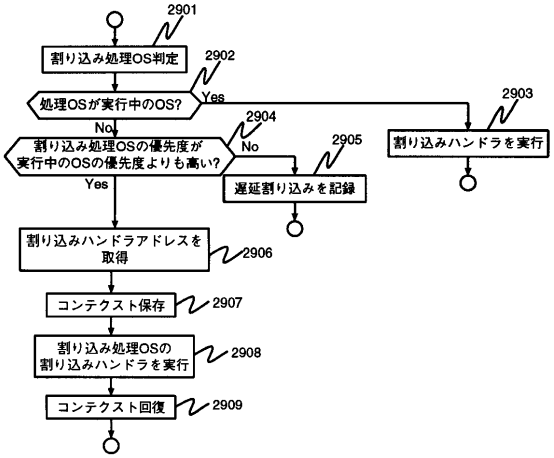
【図２８】

図２８



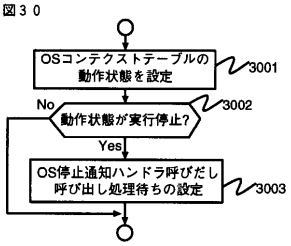
【図２９】

図２９



【図３０】

図３０



フロントページの続き

(72)発明者　金子　茂則
茨城県日立市大みか町五丁目２番１号　株式会社日立製作所　大みか工場内
(72)発明者　大野　洋
茨城県日立市大みか町五丁目２番１号　株式会社日立製作所　大みか工場内
(72)発明者　井上　太郎
神奈川県川崎市麻生区王禅寺１０９９番地　株式会社日立製作所　システム開発研究所内
(72)発明者　柴田　隆
神奈川県横浜市戸塚区戸塚町５０３０番地　株式会社日立製作所　ソフトウェア開発本部内

審査官　▲羽▼立　章二

(56)参考文献　特開平３－１０３９３１（ＪＰ，Ａ）
特開平５－２７９５４（ＪＰ，Ａ）
特開昭６４－４８３８（ＪＰ，Ａ）
特開昭６１－２０８１２５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.[7]，ＤＢ名)
G06F 9/46- 9/54
G06F 9/06
G06F 9/44- 9/445