(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2010-73214
(P2010-73214A)
(43)公開日　平成22年4月2日(2010.4.2)

(51)Int.Cl.	F I	テーマコード（参考）
G06F 9/48 (2006.01)	G06F 9/46 452Z	

審査請求 有　請求項の数 4　ＯＬ　（全 14 頁）

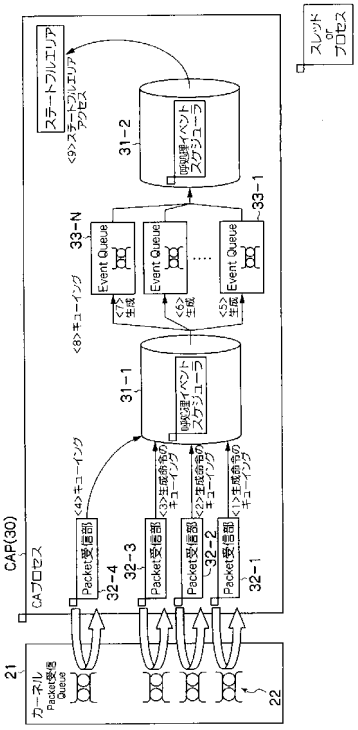
(21)出願番号　特願2009-260064 (P2009-260064)
(22)出願日　平成21年11月13日 (2009.11.13)
(62)分割の表示　特願2004-379909 (P2004-379909)の分割
原出願日　平成16年12月28日 (2004.12.28)

（特許庁注：以下のものは登録商標）
１．Ｌｉｎｕｘ

(71)出願人　000000295
沖電気工業株式会社
東京都港区西新橋三丁目１６番１１号
(74)代理人　100090620
弁理士　工藤　宣幸
(74)代理人　100161861
弁理士　若林　裕介
(72)発明者　小池　友岳
東京都港区西新橋三丁目１６番１１号　沖電気工業株式会社内
(72)発明者　安藤　智和
東京都港区西新橋三丁目１６番１１号　沖電気工業株式会社内

(54)【発明の名称】マルチ呼処理スレッド処理方法及び呼処理装置

(57)【要約】
【課題】　ＯＳのＴＳＳスケジューラの実装に依存せず、ＣＰＵを最大限に使用し、呼処理イベントの処理を実行可能とするマルチ呼処理スレッド処理方法を提供する。
【解決手段】　呼処理プログラムは、ＶｏＩＰアプリに加え、カーネル上位の呼処理イベントエンジンを有する。このエンジンにおいて、各パケット受信部は、呼制御パケットを受信すると、カーネルのＴＳＳスケジューラにキューイング制御を実行させた後、第１のスケジューラに呼処理イベントをキューイングする。第１のスケジューラは、イベント処理用のスレッドを生成し、このイベント処理スレッドにより、呼処理イベントがキューから取り出されて実行される。この実行では、各種の呼処理サービスに係る各イベントキュー管理オブジェクトが生成され、呼処理イベントを振り分けてキューイングさせる。第２のスケジューラは、イベント処理スレッドを生成し、各イベントキュー管理オブジェクトから呼処理イベントを抽出して実行させる。
【選択図】　図７



【特許請求の範囲】
【請求項１】
シングルＣＰＵがメモリ上に展開された呼処理プログラムに従い、輻輳的に発生する呼処理イベントを実行するマルチ呼処理スレッド処理方法において、
上記呼処理プログラムは、ＶｏＩＰアプリケーションに加えて呼処理イベントエンジンを有し、ＯＳに実装されているＴＳＳスケジューラを含むカーネルの上位に上記呼処理イベントエンジンを位置させると共に、上位呼処理イベントエンジンの上位に上記ＶｏＩＰアプリケーションを位置させ、
上記呼処理イベントエンジンによるコールエージェントプロセスは、１又は複数のパケット受信部と、第１の呼処理イベントスケジューラと、生成される１又は複数のイベントキュー管理オブジェクトと、第２の呼処理イベントスケジューラとを有し、
上記各パケット受信部は、呼制御に係るパケットを受信した際には、上記カーネルにその呼制御信号を引き渡して、上記ＴＳＳスケジューラにおけるキューイング制御を実行させた後、呼制御信号の返却を受けて、上記第１の呼処理イベントスケジューラに呼処理イベントをキューイングし、再びパケットの入力を待つように処理を戻し、
上記第１の呼処理イベントスケジューラは、呼処理イベントが入力されると、イベント処理を実行するためのスレッドを生成し、このイベント処理スレッドにより、呼処理イベントがキューから取り出されてイベント処理が実行され、
上記イベント処理の実行では、各種の呼処理サービスに係るそれぞれの呼処理イベントをキューイングしておくための上記各イベントキュー管理オブジェクトが生成され、呼処理イベントを振り分け、生成した上記各イベントキュー管理オブジェクト内のキュー配列へキューイングし、
上記第２の呼処理イベントスケジューラは、イベント処理スレッドを生成し、グルーピングされた上記各イベントキュー管理オブジェクトから呼処理イベントを抽出し、この抽出の際には、イベントキュー管理オブジェクトとイベント処理スレッドをバインドさせ、ステートフルエリアをアクセスしながら呼処理イベントを実行させる
ことを特徴とするマルチ呼処理スレッド処理方法。
【請求項２】
上記第１及び第２の呼処理イベントスケジューラが、１又は複数の処理時間が短い呼処理イベントを、上記ＴＳＳスケジューラが呼処理イベントの１回の処理に割り当てている最大のＣＰＵ時間を使用して処理させることを特徴とする請求項１に記載のマルチ呼処理スレッド処理方法。
【請求項３】
シングルＣＰＵがメモリ上に展開された呼処理プログラムに従い、輻輳的に発生する呼処理イベントを実行する呼処理装置において、
上記呼処理プログラムは、ＶｏＩＰアプリケーションに加えて呼処理イベントエンジンを有し、ＯＳに実装されているＴＳＳスケジューラを含むカーネルの上位に上記呼処理イベントエンジンを位置させると共に、上位呼処理イベントエンジンの上位に上記ＶｏＩＰアプリケーションを位置させ、
上記呼処理イベントエンジンによるコールエージェントプロセスは、１又は複数のパケット受信部と、第１の呼処理イベントスケジューラと、生成される１又は複数のイベントキュー管理オブジェクトと、第２の呼処理イベントスケジューラとを有し、
上記各パケット受信部は、呼制御に係るパケットを受信した際には、上記カーネルにその呼制御信号を引き渡して、上記ＴＳＳスケジューラにおけるキューイング制御を実行させた後、呼制御信号の返却を受けて、上記第１の呼処理イベントスケジューラに呼処理イベントをキューイングし、再びパケットの入力を待つように処理を戻し、
上記第１の呼処理イベントスケジューラは、呼処理イベントが入力されると、イベント処理を実行するためのスレッドを生成し、このイベント処理スレッドにより、呼処理イベントがキューから取り出されてイベント処理が実行され、
上記イベント処理の実行では、各種の呼処理サービスに係るそれぞれの呼処理イベント

をキューイングしておくための上記各イベントキュー管理オブジェクトが生成され、呼処理イベントを振り分け、生成した上記各イベントキュー管理オブジェクト内のキュー配列へキューイングし、
上記第２の呼処理イベントスケジューラは、イベント処理スレッドを生成し、グルーピングされた上記各イベントキュー管理オブジェクトから呼処理イベントを抽出し、この抽出の際には、イベントキュー管理オブジェクトとイベント処理スレッドをバインドさせ、ステートフルエリアをアクセスしながら呼処理イベントを実行させる
ことを特徴とする呼処理装置。
【請求項４】
上記第１及び第２の呼処理イベントスケジューラが、１又は複数の処理時間が短い呼処理イベントを、上記ＴＳＳスケジューラが呼処理イベントの１回の処理に割り当てている最大のＣＰＵ時間を使用して処理させることを特徴とする請求項３に記載の呼処理装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明はマルチ呼処理スレッド処理方法及び呼処理装置に関し、例えば、ＴＳＳ（タイム・シェアリング・システム）スケジューラを搭載した汎用マルチタスクＯＳを適用したＶｏＩＰサーバに適用し得るものである。
【背景技術】
【０００２】
多量のＶｏＩＰ信号を取り扱うＶｏＩＰサーバにおいては、ＶｏＩＰソフトスイッチが適用されている。このＶｏＩＰソフトスイッチが適用される、複数のユーザから同時にＶｏＩＰ信号が到来するシステムでは、Ｌｉｎｕｘ（ＵＮＩＸ；登録商標）・Ｗｉｎｄｏｗｓ（登録商標）等の汎用マルチタスクＯＳを用い、シングルプロセッサ環境下でも、複数のプロセス（ユーザによる呼処理要求）があった場合に、ＣＰＵ使用時間を分割して分配することで、あたかも同時に処理しているかのように見せるＴＳＳスケジューラが採用されている。ＴＳＳスケジューラについては、各種文献に記載されている（例えば、非特許文献１参照）。
【０００３】
また、現行のハードウェアやオペレーションシステムは、このＴＳＳスケジューラによるコンピューティングを行った際に高速に動作するよう最適化されている。
【先行技術文献】
【非特許文献】
【０００４】
【非特許文献１】澤田勉、澤田綾子、永井正武共著、「ＬｉｎｕｘとＷｉｎｄｏｗｓを理解するためのＯＳ入門」、共立出版株式会社２００３年１１月発行、１２６～１３０頁
【発明の概要】
【発明が解決しようとする課題】
【０００５】
ＴＳＳスケジューラはクォンタム（一定の時間）をプロセスに割り当てて実行している。通常、クォンタムの値は８～１０ｍ秒程度となっている。この時間単位で、処理中のプロセスがＣＰＵの処理から外され、別のプロセスへＣＰＵを渡す。このとき、処理中のプロセスに係るＣＰＵキャッシュはフラッシュされ、再度、別のプロセスに係るキャッシュが構築される。
【０００６】
しかし、一般的な呼制御信号のそれぞれは非常に短いパケットで構成されており、それぞれのイベントを処理するために使用する時間は、クォンタムより相当短い時間であり、そのような短い時間で処理が完了してしまう。これに合わせて、ＯＳのチューニングなどによりクォンタムを短く設定することができるが、ディスパッチャによるＣＰＵキャッシュのフラッシュが多発することとなり、ＣＰＵの性能が著しく低下することとなる。

【０００７】
さらに、ステートフルなＶｏＩＰ呼制御を実現する場合は、イベントそれぞれが関連性を持ち、イベント処理が各状態に左右される。このときの処理の流れを一般的なマルチスレッド環境下で実現した呼制御アプリケーション（ＶｏＩＰ呼制御プロセス）例を、図２を用いて説明する。
【０００８】
呼信号（パケット）が入力されると、カーネル（のスケジューラ）内でのキューイングによる待ち時間の後、各入力部（Ｐａｃｋｅｔ受信部）から呼処理イベントキューオブジェクトが生成され、イベントがキューイングされる（＜１＞、＜２＞、＜３＞）。呼処理イベントキュー毎にスレッドを割り当て、イベントドリブンでスレッドを生成若しくは予め生成済みのスレッドを割り当てる。このイベントキューは、呼グループ毎に生成されるオブジェクトとなっており、既存のオブジェクトへのイベントが入力された場合は該当するオブジェクトへのキューイングのみの動作となる（＜４＞）。このイベントドリブンでスレッドを生成する方式はそれぞれのイベントにスレッドが割り当てられるため、イベント処理のレスポンス時間の短縮が望める。しかし、呼イベント処理時間がシステムのスケジューラのクォンタムより短い場合、他のスレッド動作にＣＰＵを渡すために無駄なコンテキストスイッチが必要となる。システムによっては他スレッドへＣＰＵを迅速に渡すことができない場合もあり、ＣＰＵによるユーザプログラムの実行可能時間を十分に使えない。
【０００９】
次に、それぞれの呼オブジェクトは呼処理を実行する中でステートフルデータエリアである呼処理のリソースを生成（＜５＞）、変更（＜６＞）又は閲覧（＜７＞）する動作が頻繁に発生する。これらの動作は複数のオブジェクトがひとつのステートフルデータへアクセスすることを示している。このとき、スレッドの再入により、データの整合性が破壊されないため、データに対する排他制御を実現する必要がある。これは同時にスレッドが同一のエリアに対してアクセスを行った場合、唯一のスレッドのみが動作し、他のスレッドの動作を停止させるという制御となる。最も簡易的であり高速な排他制御を実現するために、ほぼ全てのＯＳが機能提供しているＭｕｔｅｘ（相互排他ロック）等の排他プリミティブがあるが、これを使用した排他制御では、他のスレッドにＣＰＵが渡されるまで、カーネル内のスケジューラによるディスパッチャ処理に依存し、結果として大幅なスループット低減につながる。
【００１０】
また、イベントドリブンに複数のスレッドを生成することにより、システムのＴＳＳスケジューラにてサービスインタラクティブ性を向上することができる一方、各呼イベント実行契機の発生時間の予測は不可能である。
【００１１】
図３は、一般的なＴＳＳスケジューラにて輻輳的なイベント処理を行った場合の実行タイミングについてのタイムテーブル例を示している。図３の例は、簡易的に、１プロセッサシステムでイベント処理以外の別処理が極力ないものを想定した。
【００１２】
各イベント処理が終了すると、他プロセスにＣＰＵを渡すディスパッチ（ＤＰ）が発生する。また、クォンタムＱをオーバして処理を続けないように強制的に割り込みが発生しディスパッチを挟んでイベント処理の切り替えが発生する（図３ではイベント処理＜２＞から＜３＞）。これにより、不用意なディスパッチを招くこととなる。上述した通り、ディスパッチが発生すると、ＣＰＵ内のキャッシュを再構築するため、ユーザプロセス処理のレイテンシー（呼び出し）に繋がる。また、スレッドごとの優先制御を行わないと、イベント処理＜２＞が処理途中でディスパッチされた際、別のイベント処理＜３＞が割り込んで順序性を乱してしまうこととなる。
【００１３】
そのため、ＯＳのＴＳＳスケジューラの実装に依存せず、ＣＰＵを最大限に使用し、呼

処理イベントの処理を実行することが可能なマルチ呼処理スレッド処理方法及び呼処理装置が望まれている。

【課題を解決するための手段】

【００１４】

かかる課題を解決するため、第１の本発明は、シングルＣＰＵがメモリ上に展開された呼処理プログラムに従い、輻輳的に発生する呼処理イベントを実行するマルチ呼処理スレッド処理方法において、（１）上記呼処理プログラムは、ＶｏＩＰアプリケーションに加えて呼処理イベントエンジンを有し、ＯＳに実装されているＴＳＳスケジューラを含むカーネルの上位に上記呼処理イベントエンジンを位置させると共に、上位呼処理イベントエンジンの上位に上記ＶｏＩＰアプリケーションを位置させ、（２）上記呼処理イベントエンジンによるコールエージェントプロセスは、１又は複数のパケット受信部と、第１の呼処理イベントスケジューラと、生成される１又は複数のイベントキュー管理オブジェクトと、第２の呼処理イベントスケジューラとを有し、（３）上記各パケット受信部は、呼制御に係るパケットを受信した際には、上記カーネルにその呼制御信号を引き渡して、上記ＴＳＳスケジューラにおけるキューイング制御を実行させた後、呼制御信号の返却を受けて、上記第１の呼処理イベントスケジューラに呼処理イベントをキューイングし、再びパケットの入力を待つように処理を戻し、（４）上記第１の呼処理イベントスケジューラは、呼処理イベントが入力されると、イベント処理を実行するためのスレッドを生成し、このイベント処理スレッドにより、呼処理イベントがキューから取り出されてイベント処理が実行され、（５）各種の呼処理サービスに係るそれぞれの呼処理イベントをキューイングしておくための上記各イベントキュー管理オブジェクトが生成され、呼処理イベントを振り分け、生成した上記各イベントキュー管理オブジェクト内のキュー配列へキューイングし、（６）上記第２の呼処理イベントスケジューラは、イベント処理スレッドを生成し、グルーピングされた上記各イベントキュー管理オブジェクトから呼処理イベントを抽出し、この抽出の際には、イベントキュー管理オブジェクトとイベント処理スレッドをバインドさせ、ステートフルエリアをアクセスしながら呼処理イベントを実行させることを特徴とする。

【００１５】

第２の本発明は、シングルＣＰＵがメモリ上に展開された呼処理プログラムに従い、輻輳的に発生する呼処理イベントを実行する呼処理装置において、（１）上記呼処理プログラムは、ＶｏＩＰアプリケーションに加えて呼処理イベントエンジンを有し、ＯＳに実装されているＴＳＳスケジューラを含むカーネルの上位に上記呼処理イベントエンジンを位置させると共に、上位呼処理イベントエンジンの上位に上記ＶｏＩＰアプリケーションを位置させ、（２）上記呼処理イベントエンジンによるコールエージェントプロセスは、１又は複数のパケット受信部と、第１の呼処理イベントスケジューラと、生成される１又は複数のイベントキュー管理オブジェクトと、第２の呼処理イベントスケジューラとを有し、（３）上記各パケット受信部は、呼制御に係るパケットを受信した際には、上記カーネルにその呼制御信号を引き渡して、上記ＴＳＳスケジューラにおけるキューイング制御を実行させた後、呼制御信号の返却を受けて、上記第１の呼処理イベントスケジューラに呼処理イベントをキューイングし、再びパケットの入力を待つように処理を戻し、（４）上記第１の呼処理イベントスケジューラは、呼処理イベントが入力されると、イベント処理を実行するためのスレッドを生成し、このイベント処理スレッドにより、呼処理イベントがキューから取り出されてイベント処理が実行され、（５）上記イベント処理の実行では、各種の呼処理サービスに係るそれぞれの呼処理イベントをキューイングしておくための上記各イベントキュー管理オブジェクトが生成され、呼処理イベントを振り分け、生成した上記各イベントキュー管理オブジェクト内のキュー配列へキューイングし、（６）上記第２の呼処理イベントスケジューラは、イベント処理スレッドを生成し、グルーピングされた上記各イベントキュー管理オブジェクトから呼処理イベントを抽出し、この抽出の際には、イベントキュー管理オブジェクトとイベント処理スレッドをバインドさせ、ステートフルエリアをアクセスしながら呼処理イベントを実行させることを特徴とする。

【発明の効果】
【００１６】
本発明のマルチ呼処理スレッド処理方法及び呼処理装置によれば、ＯＳのＴＳＳスケジューラの実装に依存せず、ＣＰＵを最大限に使用し、呼処理イベントの処理を実行することができるようになる。
【図面の簡単な説明】
【００１７】
【図１】実施形態のマルチ呼処理スレッド処理方法の実行環境の説明図である。
【図２】ステートフルなＶｏＩＰ呼制御を実現する従来のＶｏＩＰ呼制御プロセス例を示す説明図である。
【図３】一般的なＴＳＳスケジューラにて輻輳的なイベント処理を行った場合の実行タイミングについてのタイムテーブル例を示す説明図である。
【図４】実施形態の呼処理イベントスケジューラをＴＳＳスケジューラに併用した場合における輻輳的なイベント処理を行った場合の実行タイミングについてのタイムテーブル例を示す説明図である。
【図５】実施形態のユーザ実行用スレッドの基本動作を示すフローチャートである。
【図６】実施形態のハングチェックスレッドの動作を示すフローチャートである。
【図７】実施形態の呼処理イベントエンジンによるＣＡ（コールエージェント）プロセスの構造例を示す説明図である。
【図８】実施形態の呼処理イベントスケジューラの正常シーケンスを示すシーケンス図である。
【図９】実施形態の呼処理イベントスケジューラの異常シーケンスを示すシーケンス図である。
【発明を実施するための形態】
【００１８】
（Ａ）実施形態
以下、本発明によるマルチ呼処理スレッド処理方法及び呼処理装置の一実施形態を、図面を参照しながら詳述する。
【００１９】
この実施形態のマルチ呼処理スレッド処理方法が実装されたマルチ呼処理スレッド処理装置（呼処理装置）は、例えば、多量のＶｏＩＰ信号（例えば、８０万回線）を取り扱うコールエージェント（電話交換システムの一種）に適用されているものであり、ハードウェア的には、一般的なコールエージェントと同様である。すなわち、ＣＰＵ、メモリ、内蔵ＨＤＤ等の大容量記憶装置、キーボード、マウス、ディスプレイ、通信インタフェース部などを備えており、ＣＰＵは、システムバスを介してメモリに接続され、また、システムバス及び入出力デバイスを介して、大容量記憶装置、キーボード、マウス、ディスプレイ、通信インタフェース部等と接続されている。
【００２０】
このマルチ呼処理スレッド処理装置は、例えば、Ｌｉｎｕｘ・Ｗｉｎｄｏｗｓ等の汎用マルチタスクＯＳを適用している。上述したように、汎用マルチタスクＯＳはカーネル内にＴＳＳスケジューラを有している。
【００２１】
マルチ呼処理スレッド処理装置１は、図１に示すように、汎用マルチタスクＯＳ２０におけるカーネル２１内のＴＳＳスケジューラ２２に加え、ユーザ層に位置する、スケジューラ機能（呼処理イベントスケジューラ３１）を有する呼処理イベントエンジン３０を有しており、スケジュール機能面からは、ＴＳＳスケジューラ２２は、イベントエンジン３０を介して、ＶｏＩＰアプリケーション４０に係るスレッドを処理対象とし得る。言い換えると、イベントスケジューラ３１は、カーネル２１のＴＳＳスケジューラ２２上で動作するコンテキストである。
【００２２】

図４は、実施形態に係るマルチ呼処理スレッド処理装置１が呼処理イベントを実行した際のタイムテーブル例を示しており、主として、呼処理イベントスケジューラ３１の処理イメージを示している。図４は、従来に係る図３に対応する図面である。

【００２３】

呼処理イベントスケジューラ３１は、キュー管理され順序制御が行われた複数の処理時間が短いイベントを繋げ、一回に割り当てられている最大のＣＰＵ時間（ＴＳＳスケジューラ２２が提供するクォンタム）Ｑを使用して呼処理イベントの処理を実施する。図４の例は、イベント処理＜１＞だけではＣＰＵ時間Ｑが埋まらず、呼処理イベントスケジューラ３１は、そのＣＰＵ時間Ｑ内でイベント処理＜２＞（の前半部分）を続けて処理させている。各イベント処理の切り替わる間には（図４の「Ｅｖ切」）、呼処理イベントスケジューラ３１の制御処理が割り込むことになるが、即座に次のイベント処理を実行させる。これにより、各イベント処理の間にディスパッチ（図４の「ＤＰ」）や呼制御処理とは関係のない別プロセス（図４の「別処理」）の処理が実行されないため、同じＣＰＵ処理時間Ｑで多くのイベントを処理することを可能としている。なお、ディスパッチは、ＴＳＳスケジューラ２２により生じるものである。

【００２４】

ここで、ディスパッチが不用意に発生しないため、ＣＰＵキャッシュを最大限に活用できることなり、さらなる高速化に繋がっている。なお、この実施形態に係る図４と従来に係る図３とは、同様なイベント処理の系列の場合であるが、従来における５回のディスパッチ回数が、この実施形態では３回に減少している。

【００２５】

以下では、この実施形態の呼処理イベントスケジューラ３１による制御動作について説明する。

【００２６】

一つのイベントがスケジューラ３１に入力されると、順序制御のための配列にキューイングされる。ここで、イベントが輻輳しているかどうかの判定をキューイングされているイベントの個数から行い、輻輳が認められた場合はこの旨をスケジューラ３１へ記録する。

【００２７】

スケジューラ３１内には最低１つの監視スレッドが生成されており、イベントキューの監視、及び、イベント処理を実行しているスレッドの監視を行っている。イベントが入力されキューイングされると、この監視スレッドにより新たなイベント処理を実施するためのスレッド（ユーザ実行用スレッド）を生成する。

【００２８】

図５は、ユーザ実行用スレッドの基本動作を示している。ユーザ実行用スレッドは、入力されたイベントに係るコンテキストを生成した後（Ｓ１０１）、イベントの有無を確認し（Ｓ１０２）、イベントが存在すると、実行するユーザ（例えば、後述する呼処理オブジェクトグループ）を選択し（Ｓ１０３）、そのユーザの処理を実行して（Ｓ１０４）、上述したイベントの有無の確認に戻る。このユーザ処理の実行中においては、適宜イベントが生成される。また、後述するハングチェックのための時間計測なども行われる。

【００２９】

ステップＳ１０２～Ｓ１０４でなるループは繰り返され、ＴＳＳスケジューラ２２の最大ＣＰＵ時間Ｑの経過によるディスパッチがあったときに、そのループ処理は中断され、ＴＳＳスケジューラ２２による自己へのディスパッチによって再開される。これにより、図４に示したような、最大ＣＰＵ時間Ｑ内でのイベント切り替えを実現している。

【００３０】

前述したキューイングする別スレッドによって輻輳状態に陥っていることが判明した場合、監視スレッドは、新たなイベント処理が生成するか、新たな監視スレッドを生成して自らイベント処理スレッドに切り替わるかを選択する。監視スレッドがイベント処理スレッドに切り替わってそのまま動作する理由は、輻輳時、新たなスレッドを生成するための

リソースを確保し、確実に動作可能なスレッドの生成を保証することが困難であるからである。また、監視スレッドの方がスレッドの優先度が高いため、より確実にイベント処理を行い、キューを刈り取ることが可能になっているためである。

【００３１】

監視スレッドによるイベント処理スレッドの監視は、各イベントの実行時間の計測によりハングチェックを行う。イベント処理スレッドは、各イベント処理の切り替わりの際にタイムスタンプをイベントスケジューラ３１へ記録する。監視スレッドは、このタイムスタンプから各イベント処理の実行にかかっている時間を計算し、閾値内に収まっているかどうかを判定する。各イベント処理内部で無限ループなどが発生した場合、閾値の時間を経過して新たなイベント処理スレッドを再生成する。単位時間内に、このイベント処理スレッドの再生成が複数発生すると、デッドロックしたものとみなし、プロセスの再開要求を行う。

【００３２】

図６は、監視スレッド（ハングチェックスレッド）のこのような動作をフローチャートで示している。監視スレッド（ハングチェックスレッド）は、ハングチェック用のコンテキストを生成した後（Ｓ２０１）、実行中ユーザ（実行中のイベント処理イベント）の有無を判定し（Ｓ２０２）、実行中ユーザが存在すれば、実行中ユーザ分だけ、実行中ユーザを選択し（Ｓ２０３）、タイムバーストしていないことを確認する（Ｓ２０４）。そして、タイムバーストした実行中ユーザがあれば、そのようなユーザに対するユーザ処理実行用のコンテキストを生成して（Ｓ２０５）、上述したステップＳ２０２に戻る。

【００３３】

上述した呼処理イベントスケジューラ３１を導入したことにより、ＯＳ２０のＴＳＳスケジューラ２２の実装に依存せず、ＣＰＵを最大限に使用し、呼処理イベントの処理を実行することが可能となる。

【００３４】

スレッド数をコントロールし、要求された呼イベント処理に必要なスレッド数のみを生成するため、ＴＳＳスケジューラ２２の環境下でも、ステートフルデータを保持する呼イベント同士のコンテキスト衝突が発生せず、高速に処理することが可能である。また、システム内のスレッド数を最小限に留めることが可能となり、カーネル２１のスケジューラ２２による不要なディスパッチ処理の発生を抑え、結果として、呼イベント実行契機のタイミングを短くすることが可能となり、サービスインタラクティブ性を確保できる。

【００３５】

この実施形態の呼処理イベントエンジン３０は、コールエージェント（電話交換システム）のような輻輳的に発生する呼処理イベントを高速実行するためのものであり、コンテキストの連結／管理と、順序制御と、呼処理コンテキストのハング（ハングアップ）チェックとを行うものである。

【００３６】

図７は、実施形態の呼処理イベントエンジン３０によるＣＡ（コールエージェント）プロセスの構造例を示す説明図であり、従来に係る上述した図２に対応する図面である。なお、従来では、カーネルの上位が直ちにＶｏＩＰアプリケーションであったが、この実施形態の場合には、カーネルの上位に呼処理イベントエンジン３０が存在し、さらに、その上にＶｏＩＰアプリケーションが存在し、状態に応じてプロセスやスレッドが適宜生成される。

【００３７】

呼処理イベントエンジン３０によるＣＡ（コールエージェント）プロセスＣＡＰは、１又は複数（図７では４個を示している）のパケット受信部３２－１～３２－４を有している。各パケット受信部３２－１～３２－４が信号を受信した際には、カーネル２１にその呼制御信号を引き渡し、カーネル２１のＴＳＳスケジューラ２２におけるキューイング制御の後、各パケット受信部３２－１～３２－４にカーネル２１からの呼制御信号が与えられる。

【００３８】
ＣＡプロセスＣＡＰは、前段に、カーネル２１からの呼制御信号を受信した１次イベントを処理するためのイベントスケジューラ３１－１を保持し、後段に、呼処理サービスを処理するためのイベントスケジューラ３１－２を保持している。また、ＣＡプロセスＣＡＰは、これら２個のイベントスケジューラ３１－１及び３１－２を繋ぐため、各種の呼処理サービスに係るそれぞれのイベントをキューイングしておくためのイベントキュー管理オブジェクト３３－１～３３－Ｎを配置する。イベントキュー管理オブジェクト３３－１～３３－Ｎの基本動作は、ＦＩＦＯ（Ｆｉｒｓｔ－Ｉｎ　Ｆｉｒｓｔ－Ｏｕｔ）である。
【００３９】
２個のイベントスケジューラ３１－１及び３１－２はそれぞれ、図４～図６を用いて説明したイベントスケジューラ３１である。
【００４０】
各パケット受信部３２－１～３２－４が信号を受信した際、前段のイベントスケジューラ３１－１へキューイングし、再びパケットの入力を待つため、即座に処理が戻る（＜１＞～＜４＞）。イベントスケジューラ３１－１は、イベントが入力されると、イベント処理を実行するためのスレッドを生成する。イベント処理スレッドにより、イベントがキューから取り出されイベント処理が実行されていく。図７は、イベントキュー管理オブジェクト３３－１～３３－Ｎを生成し、イベントを振り分け、生成したオブジェクト内のキュー配列へキューイングしていく処理を行う場合を示している（＜５＞～＜８＞）。
【００４１】
後段の呼処理を実行するためのイベントスケジューラ３１－２は、イベント処理スレッドを生成し、グルーピングされた各イベントキュー管理オブジェクト（呼処理オブジェクトグループ）３３－１～３３－Ｎからイベントを抽出し、呼処理イベントを実行させる。呼処理イベントの実行時には、適宜、ステートフルエリアがアクセスさせる。
【００４２】
イベントスケジューラ３１－２は、上述したイベントの抽出などを行う際には、イベントキュー管理オブジェクトとイベント処理スレッドをバインドさせて以降の処理順序を保つ。
【００４３】
なお、イベントキュー管理オブジェクト分、イベント処理スレッドを生成することも可能であるが、カーネル２１内のＲｕｎキューを輻輳させてしまうため、イベント処理を繋ぎスレッド数のコントロールを行うこととした。また、コントロールして生成したスレッド数を最小限に抑えることにより、上位のステートフルエリア３４へのアクセスでスレッド同士の競合が発生せず、ロックによる待機時間をなくすことができる。
【００４４】
図８は、後段の呼処理イベントスケジューラ３１－２に着目し、その正常シーケンスの場合を示しており、カーネル２１や前段の呼処理イベントスケジューラ３１－１の動作は省略している。
【００４５】
信号入力スレッド（図７でのパケット受信部に相当）３２が非同期信号を受信し、呼処理オブジェクトグループ３３－ＧＡの信号入力オブジェクトをコールする（Ｓ３０１）。このとき、呼処理オブジェクトグループ３３－ＧＡの信号入力オブジェクトは、呼処理イベントスケジューラ３１（３１－２）に、イベントを処理するためのコンテキストを要求する（Ｓ３０２）。要求がキューイングされると、要求したコンテキストが生成されるまで、全ての呼処理オブジェクトグループ３３－ＧＡ、３３－ＧＢや信号入力スレッド３２は、その加入者サービスについて対応する呼処理に対し、排他制御を実行する（Ｓ３０３）。
【００４６】
例えば、呼処理オブジェクトグループ３３－ＧＡはある発信側加入者に係る呼処理オブジェクトのグループであり、呼処理オブジェクトグループ３３－ＧＢはその呼の着信側加

入者に係る呼処理オブジェクトのグループである。

【００４７】

そして、呼処理イベントスケジューラ３１（３１－２）が要求されたコンテキストを生成し（Ｓ３０４）、呼処理オブジェクトグループ３３－ＧＡに関する呼処理を実施する（Ｓ３０５）。

【００４８】

このような呼処理オブジェクトグループ３３－ＧＡの呼処理の実施中には、ハングチェックのために、実行時時間の実時間が計測される。図８における鋸歯状の曲線が重畳されている部分が計測期間である。

【００４９】

また、このような、ある呼処理オブジェクトグループ３３－ＧＡの呼処理の実施中に他の呼処理オブジェクトグループ３３－ＧＢで実行される非同期イベントが生成されることもあり、その通知を受けた呼処理オブジェクトグループ３３－ＧＢは、キューイングした後、自グループ３３－ＧＢ用のコンテキストを呼処理イベントスケジューラ３１に要求し、その要求の受信応答が呼処理イベントスケジューラ３１から返信されると、呼処理オブジェクトグループ３３－ＧＢは、呼処理オブジェクトグループ３３－ＧＡに関する呼処理を再開させる。呼処理オブジェクトグループ３３－ＧＡは、実行中の呼処理が終了すると、呼処理イベントスケジューラ３１にそのことを通知する。図８では、説明の簡易化のために簡単に示しているが、これにより、呼処理イベントスケジューラ３１は、呼処理オブジェクトグループ３３－ＧＢが要求したコンテキストを生成して、呼処理オブジェクトグループ３３－ＧＢに関する呼処理を実施する（Ｓ３０６）。

【００５０】

カーネル２１のＴＳＳスケジューラ２２が提供するクォンタムＱが経過すると、割り込みが発生し、呼処理オブジェクトグループ３３－ＧＢに関する呼処理が中断する（Ｓ３０７）。

【００５１】

図９は、図８の正常シーケンスに対応する異常シーケンスを示しており、図８におけるステップＳ３０４の呼処理オブジェクトグループ３３－ＧＡが要求したコンテキストを生成した以降の流れを示している。

【００５２】

呼処理オブジェクトグループ３３－ＧＡに関する呼処理の実行中において、論理矛盾によるループやデッドロックによるプログラムの停止などが生じると（Ｓ４０１）、呼処理イベントスケジューラ３１による計測時間が閾値時間を越えてしまう。

【００５３】

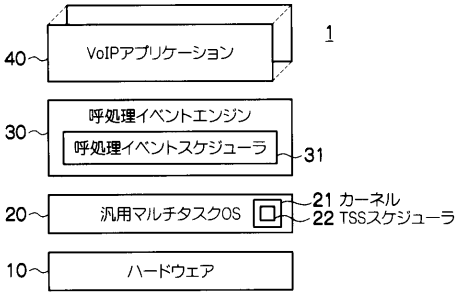
このような異常状態は、ハングチェックスレッド５０が、処理中のクォンタムＱ内で異常状態を検出するか否かの相違により、以下の第１又は第２の方法で処理される。

【００５４】

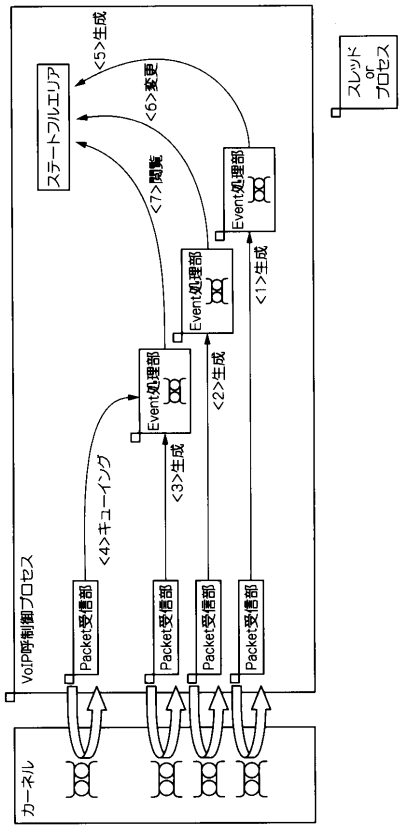
第１の方法は、以下の通りである。呼処理イベントエンジン３０におけるハングチェックスレッド５０は、種々の実行イベントに対する実行時間を監視しており（Ｓ４０２）、呼処理オブジェクトグループ３３－ＧＡに関する呼処理イベントの実行時間が上述の原因などによって閾値時間を越えてタイムアウトしたことを検出する（Ｓ４０３）。このとき、ハングチェックスレッド５０は、新たな呼処理イベントスケジューラ３１ＮＥＷを生成させる（Ｓ４０４）。呼処理イベントスケジューラ３１ＮＥＷは、例えば、呼処理オブジェクトグループ３３－ＧＢに関するコンテキストを生成し（Ｓ４０４）、呼処理オブジェクトグループ３３－ＧＢに関するコンテキストを生成して呼処理を実施する（Ｓ４０５、Ｓ４０６）。以上のような新たな呼処理イベントスケジューラ３１ＮＥＷの生成により、呼処理イベントスケジューラによる処理が継続したように見える。このような呼処理イベントスケジューラ３１ＮＥＷに従う呼処理の実行中においても、カーネル２１のＴＳＳスケジューラ２２が提供するクォンタムＱが経過すると、割り込みが発生し、呼処理オブジェクトグループ３３－ＧＢに関する呼処理が中断する（Ｓ４０７）。

【００５５】
第２の方法は、以下の通りである。ハングチェックスレッド５０の実行時間の監視タイミングの前に、カーネル２１のＴＳＳスケジューラ２２が提供するクォンタムＱが経過すると、割り込みが発生し（Ｓ４１０）、処理が中断する。
【００５６】
その後、カーネル２１のＴＳＳスケジューラ２２のクォンタムＱが再び、中断した処理に係るものとなって処理を復旧（再開）すると、ハングチェックスレッド５０はコンテキストの継続判定を行う（Ｓ４１１）。ここで、輻輳状態と判定されると、そのことが呼処理イベントスケジューラ３１１に通知され、ステップＳ３０４で生成したコンテキストを消滅させる（Ｓ４１２）。このようなコンテキストの消滅は、輻輳状態などのスレッドを生成した方が良いと判断された場合に行われる。
【００５７】
実施形態の呼処理イベントエンジン３０によれば、図４～図６を用いて説明した呼処理イベントスケジューラ３１を適用した効果に加え、コールエージェント（電話交換システム）のようなステートフルエリアを保持しなければならないシステムで課題であったコンテキストの競合を最小限に抑えることを、サービス形態に依存せずに、容易に達成することができる。また、システム全体で生成されるスレッドの数を最小限に抑えることが可能であり、パケット受信部などの短い処理を実施するスレッドへも頻繁に実行契機が与えられ、インタラクティブ性の向上が図ることができる。
【００５８】
（Ｂ）他の実施形態
本発明によるマルチ呼処理スレッド処理装置及び方法は、その適用対象が、上記実施形態のようなコールエージェントに限定されるものではなく、メディアゲートウェイコントローラ（ＭＧＣ）などの他の電話交換装置にも適用できる。
【符号の説明】
【００５９】
１…マルチ呼処理スレッド処理装置、１０…ハードウェア、２０…汎用マルチタスクＯＳ、２１…カーネル、２２…ＴＳＳスケジューラ、３０…呼処理イベントエンジン、３１…呼処理イベントスケジューラ、４０…ＶｏＩＰアプリケーション。

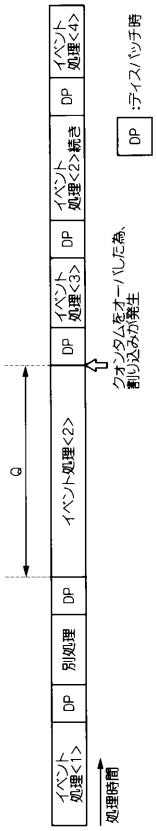
【図１】


1
40
VoIPアプリケーション
30
呼処理イベントエンジン
呼処理イベントスケジューラ
31
20
汎用マルチタスクOS
21 カーネル
22 TSSスケジューラ
10
ハードウェア

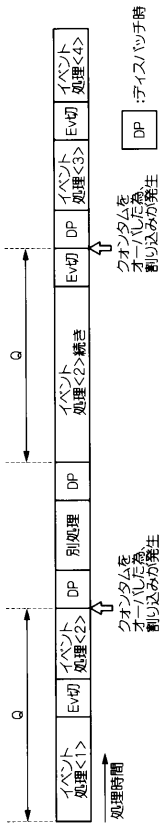
【図２】


スレッド or プロセス
ステートフルエリア
<5>生成
<6>変更
<7>開放
Event処理部
<1>生成
<2>生成
<3>生成
<4>キューイング
Packet受信部
VoIP呼制御プロセス
カーネル

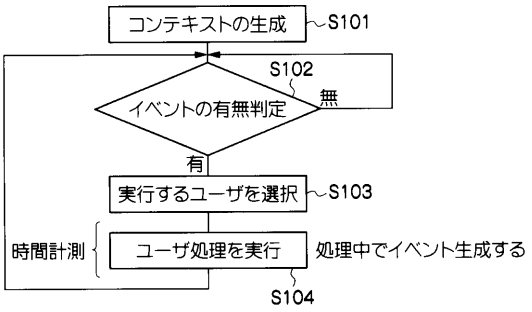
【図３】


イベント処理<1>
DP
別処理
DP
イベント処理<2>
Q
DP
イベント処理<3>
DP
イベント処理<2>続き
DP
イベント処理<4>
クオンタムをオーバした為、割り込みが発生
DP :ディスパッチ時
処理時間

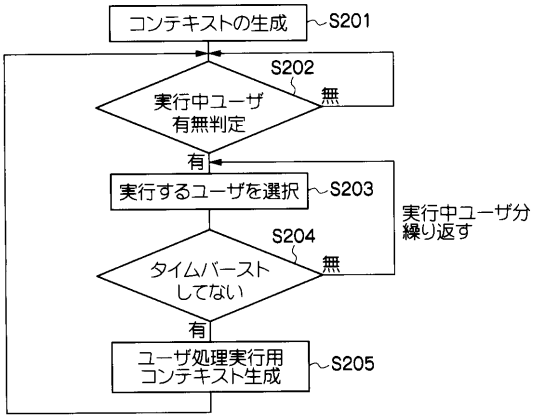
【図４】


イベント処理<1>
Ev切
イベント処理<2>
DP
別処理
DP
イベント処理<2>続き
Q
Ev切
DP
イベント処理<3>
Ev切
イベント処理<4>
クオンタムをオーバした為、割り込みが発生
クオンタムをオーバした為、割り込みが発生
DP :ディスパッチ時
処理時間

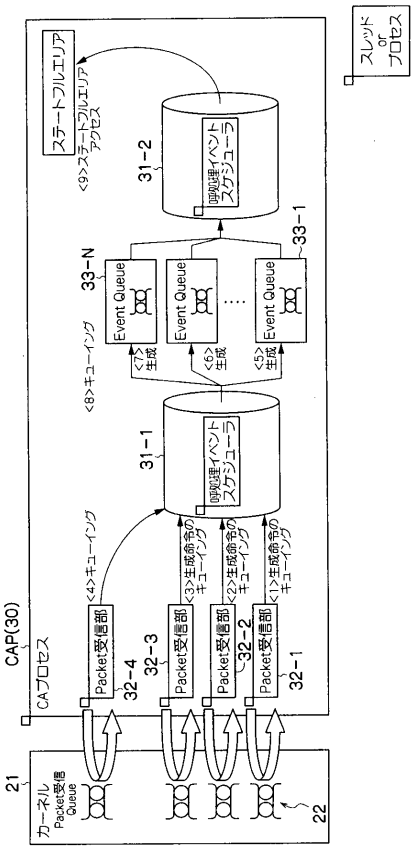
【図５】


コンテキストの生成
S101
S102
イベントの有無判定
無
有
実行するユーザを選択
S103
時間計測
ユーザ処理を実行
処理中でイベント生成する
S104

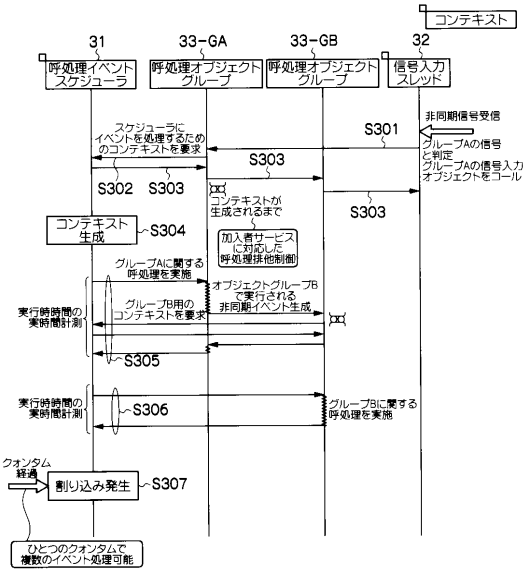
【図６】


コンテキストの生成
S201
S202
実行中ユーザ
有無判定
無
有
実行するユーザを選択
S203
実行中ユーザ分
繰り返す
S204
タイムバースト
してない
無
有
ユーザ処理実行用
コンテキスト生成
S205

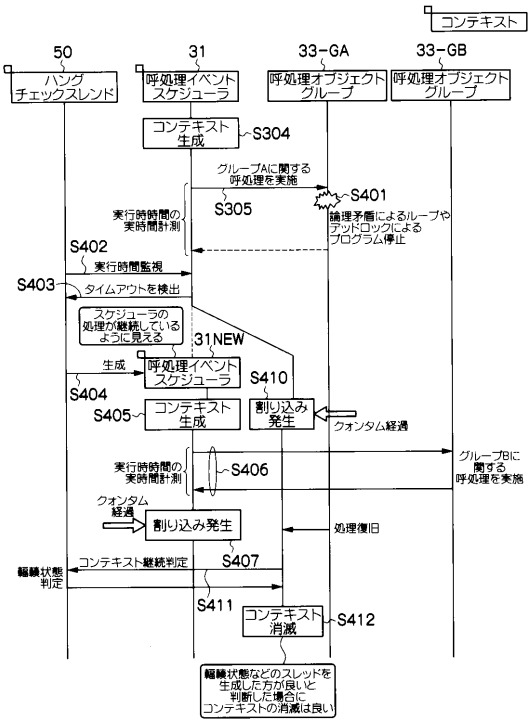
【図７】


スレッド or プロセス
ステートフルエリア
<9>ステートフルエリアアクセス
31-2
呼処理イベントスケジューラ
33-1
33-N
Event Queue
<8>キューイング
<7>生成
<6>生成
<5>生成
31-1
呼処理イベントスケジューラ
<4>キューイング
<3>生成命令のキューイング
<2>生成命令のキューイング
<1>生成命令のキューイング
CAP(30)
CAプロセス
Packet受信部
32-4
32-3
32-2
32-1
21
カーネル Packet受信 Queue
22

【図８】


コンテキスト
31
呼処理イベントスケジューラ
33-GA
呼処理オブジェクトグループ
33-GB
呼処理オブジェクトグループ
32
信号入力スレッド
非同期信号受信
S301
グループAの信号と判定
グループAの信号入力オブジェクトをコール
スケジューラにイベントを処理するためのコンテキストを要求
S303
S302
S303
コンテキストが生成されるまで
S303
コンテキスト生成
S304
加入者サービスに対応した呼処理排他制御
グループAに関する呼処理を実施
オブジェクトグループBで実行される非同期イベント生成
実行時時間の実時間計測
グループB用のコンテキストを要求
S305
実行時時間の実時間計測
S306
グループBに関する呼処理を実施
クオンタム経過
割り込み発生
S307
ひとつのクオンタムで複数のイベント処理可能

【図９】


コンテキスト
50
ハング
チェックスレッド
31
呼処理イベント
スケジューラ
33-GA
呼処理オブジェクト
グループ
33-GB
呼処理オブジェクト
グループ
コンテキスト
生成
S304
グループAに関する
呼処理を実施
S305
S401
論理矛盾によるループや
デッドロックによる
プログラム停止
実行時間の
実時間計測
S402
実行時間監視
S403
タイムアウトを検出
スケジューラの
処理が継続している
ように見える
31,NEW
生成
呼処理イベント
スケジューラ
S410
S404
S405
コンテキスト
生成
割り込み
発生
クオンタム経過
グループBに
関する
呼処理を実施
実行時間の
実時間計測
S406
クオンタム
経過
割り込み発生
処理復旧
コンテキスト継続判定
S407
輻輳状態
判定
S411
コンテキスト
消滅
S412
輻輳状態などのスレッドを
生成した方が良いと
判断した場合に
コンテキストの消滅は良い