(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3724075号
(P3724075)
(45)発行日　平成17年12月7日(2005.12.7)　(24)登録日　平成17年9月30日(2005.9.30)

(51)Int.Cl.[7]　F I
G06F　9/52　G06F　9/46　472Z

請求項の数 2　(全 13 頁)

(21)出願番号　特願平8-226404
(22)出願日　平成8年8月28日(1996.8.28)
(65)公開番号　特開平10-69392
(43)公開日　平成10年3月10日(1998.3.10)
審査請求日　平成13年3月9日(2001.3.9)

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100075096
弁理士　作田　康夫
(72)発明者　中原　雅彦
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　岩嵜　正明
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】プロセスの実行方法

(57)【特許請求の範囲】
【請求項１】
複数のプロセスが並列に動作する計算機システムにおいて、１つの共有資源を獲得し、当該資源を操作し、当該資源を解放する前記複数の中の１つのプロセスの実行方法において、
前記１つのプロセスをアボート禁止にするとともにプリエンプト禁止にした後、当該プロセスが使用する前記共有資源を獲得し、
当該プロセスのプリエンプト禁止を解除するとともに前記共有資源の操作を行なった後、当該プロセスをプリエンプト禁止にし、
当該プロセスが前記共有資源を解放した後、当該プロセスのプリエンプト禁止を解除するとともにアボート禁止を解除し、さらに、
アボート禁止中に発生した当該プロセスの強制終了要求をアボート禁止解除後の実行することを特徴とするプロセスの実行方法。
【請求項２】
請求項１記載のプロセスの実行方法において、
前記共有資源の操作を要求するプロセスのキューを設け、
前記共有資源に対する処理を要求する前記キューの先頭のプロセスをマルチプロセシング環境で実行し、
前記プロセスの実行完了後、当該プロセスに関する処理を周期的に駆動し、前記キューに登録されたプロセスを順次実行することを特徴とするプロセスの実行方法。

【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は計算機を制御するオペレーティング・システムにおけるプロセスの実行方法に係り、特に、マルチプロセシング環境を提供するオペレーティング・システムにおける共有資源を使用するプロセスの実行方法に関する。
【０００２】
【従来の技術】
複数のプロセス（またはタスク）を時分割で並列に実行するマルチプロセシング環境を提供するオペレーティング・システムでは、プロセス間で共有する資源（共有資源）が複数のプロセスに同時に割り当てられないように排他制御を行ないながらプロセスを実行する必要がある。この排他制御を実現するために、オペレーティング・システム（ＯＳ）においてロック制御と呼ばれる手法が広く使用されている。
【０００３】
ロック制御では、共有資源毎に対応する共有資源が使用中であることを示すフラグを設ける。プロセスが共有資源に対する処理を行う際には、このフラグが他のプロセスによりセットされているかを調べる。そして、他のプロセスによりフラグがセットされた状態でなければ、このプロセスによりフラグをセットして資源を占有する（これを「ロック」すると言う）。そして、この共有資源に対する処理の終了とともにプロセスがフラグをリセットして資源を解放する（これを「アンロック」すると言う）。一方、他のプロセスによりフラグがセットされた状態ならば、ＯＳはこの共有資源がアンロックされるまで、このプロセスを待ち状態にする。
【０００４】
ところで、並列に実行される優先度の低いプロセス１と優先度の高いプロセス２が存在し、二つのプロセスが共有する資源Ａがあるとする。いま、プロセス１がCPU占有中に共有資源Ａをロックし、共有資源Ａをロックしたまま休眠状態に入ったとする。この場合、CPUの使用権が割り当てられたプロセス２が共有資源Ａを使用しようとしても、共有資源Ａが他のプロセスによりロックされているため、プロセス２は共有資源Ａを使用できない。このためプロセス２は、実行継続不能となる。
【０００５】
このように優先度の低いプロセスが獲得しているロックのために、優先度の高いプロセスが実行継続不能となることを優先度逆転問題という。従来のオペレーティング・システムでは、優先度逆転問題が発生した場合は、ロックを取ったまま休眠している優先度の低いプロセスを最優先で実行させることにより、優先度の低いプロセスが獲得している資源を解放させ、優先度の高いプロセスをできるだけ早く実行できるようにしている。
【０００６】
【発明が解決しようとする課題】
連続メディア処理では、動画データなどのデータを処理するプロセスの動作に周期性が生ずる。この性質を利用したプロセス・スケジューリング方法を特願平８－７３６７３で提供した。
【０００７】
しかし、優先度逆転問題は周期性を利用したプロセス・スケジューリング方法に対しても悪影響を及ぼす。例えば、連続メディア処理を行うプロセス１、プロセス２が存在し、二つのプロセスの間で共有する資源Ａがあるとする。プロセス１とプロセス２がこの共有資源Ａをロックを使って排他制御する場合、優先度逆転問題を発生させる可能性があり、発生すればプロセスの周期駆動性を損なうことになる。
【０００８】
更に問題なのは、優先度逆転問題が発生すると、その原因となったロックを持っているプロセスが最優先にスケジュールされるため、そのロックとは関係なく連続メディア処理を行っている別のプロセスの周期駆動性にまで影響を与えることである。

【０００９】
そこで、連続メディア処理を行うシステムでは、優先度逆転問題を回避するために最初から一切ロックを用いずに計算機システムを構築する方法が考えられる。そのような計算機システムにおいて排他制御を行う方法として、共有資源使用中はプロセスにCPUを占有させる、すなわち、共有資源使用中は他のプロセスがスケジューリングされないプロセス制御方法が考えられる。これは、特願平８－９７９９７の発明で提供したプリエンプト制御方法を使用することで実現可能である。プリエンプト制御方法とは、実行中プロセスの実行が一時中断される（これを「プリエンプト」されると言う）ことを防ぐために「プリエンプト禁止」及び「プリエンプト禁止解除」のインタフェースを用意し、プリエンプト禁止にされてからプリエンプト禁止が解除されるまでの間（これを「プリエンプト禁止区間」と呼ぶ）は他のプロセスのスケジューリング要求が発行されても、実行中プロセスはプリエンプトされずに実行を続ける方法である。このような制御を行うことによって、共有資源を使用しているプロセスは常に１個であることが保証できるため、排他制御が実現できる。
【００１０】
しかし、この方法では共有資源使用中のプロセス走行区間が長時間に渡る場合には他のプロセスの周期駆動性を損なってしまう。
【００１１】
例えば、システム内で共有している使用可能な資源をキューイングするフリーリストから資源を取り出し、これを初期化して、あるプロセスの資源割当てリストに繋ぐ処理を考える。ここで排他制御が必要な共有資源となるのは、フリーリストと資源割当てリストである。
【００１２】
この処理をプリエンプト禁止区間の設定だけで行う場合、プリエンプト禁止区間は「フリーリストに触る前」から「資源割当てリストに資源を繋ぎおわる」まで設定する必要がある。すなわち、
（１）プリエンプト禁止の設定を行う
（２）フリーリストから資源を取り出す
（３）資源の初期化を行う
（４）資源をプロセスの資源割当てリストに繋ぐ
（５）プリエンプト禁止の設定を解除する
となる。ここで、資源の初期化に要する時間は短いとは限らない。実際、オペレーティング・システム内でメモリを割り当てる際のメモリの初期化等では数十ｍｓを要する。そのような場合に上記の様にプリエンプト禁止区間の設定だけで排他制御を行うと、一つのプロセスが長時間ＣＰＵを占有することになる。この結果、リアルタイム応答性が悪くなり、連続メディア処理の周期駆動性に影響を与えることになる。
【００１３】
そこで、上記処理を
（１）プリエンプト禁止の設定を行う
（２）フリーリストから資源を取り出す
（３）プリエンプト禁止の設定を解除する
（４）資源の初期化を行う
（５）プリエンプト禁止の設定を行う
（６）資源をプロセスの資源割当てリストに繋ぐ
（７）プリエンプト禁止の設定を解除する
という形に変更することが考えられる。このようにすれば、排他制御という点でのみ見れば、必要条件を満たしている。しかも、プリエンプト禁止区間の長さは１０μｓ程度に抑えられ、（４）の処理中はＣＰＵ使用権を他のプロセスに移すことができるので、リアルタイム応答性が良くなる。ところが、（４）の処理をしている途中に、この処理を行っているプロセスが外部から強制終了させられた場合に問題が生ずる。

【００１４】
一般の計算機システムでは、プログラムの暴走に備えて、外部からプロセスを強制的に停止・消滅させる機能を備えている。この機能を使って上記処理を行っているプロセスが他のプロセス等から強制終了させられた場合、（４）で初期化している資源がどの管理リストにも属さない所属不明の資源となる可能性がある。この結果、オペレーティング・システムがプロセスを消滅させる際に資源の回収ができず、その資源は永久に使用できなくなる可能性がある。これを回避するためには、フリーリスト、あるいは資源割り当てリストに資源を繋いだ状態で資源を初期化する方法が考えられるが、この方法では、前の例の処理全体をプリエンプト禁止区間で覆う方法とプリエンプト禁止区間の長さがほとんど変わらなくなる。別の方法として、初期化中の資源を管理するリストを別に用意し、（４）の処理中はこのリストに繋ぐ方法もあるが、リストの繋ぎ換え処理が多くなるため、初期化処理のオーバヘッドが大きくなる。
【００１５】
また、共有資源獲得のために必要な時間を短時間に抑えることができない場合にも、上記の方法は使用できない。
【００１６】
そこで本発明者は、プリエンプト制御方法を利用して、連続メディア処理の周期駆動性に影響を与えずに共有資源の処理を行うことができるプロセス実行方法を発明した。
【００１７】
したがって、本発明の目的は連続メディア処理を行う計算機システムの共有資源を使用するプロセスの実行方法を提供することにある。
【００１８】
【課題を解決するための手段】
本発明では、プロセスが共有資源を使用する際には、予め自プロセスが強制終了させられない（これを「アボート」されないと言い、アボートされないようにすることを「アボート禁止」にすると言う）ように宣言を行う。その上で，自プロセスの処理が中断されないように「プリエンプト禁止」にする。そして、共有資源の使用が終了した際にはプリエンプト禁止の解除を行う。共有資源に対するすべての処理が終了したら、自プロセスが強制終了させられてもよい（これを「アボート禁止解除」と呼ぶ）ことを宣言する。このアボート禁止を宣言してからアボート禁止解除を宣言するまでのプロセスが強制終了させられない区間のことをアボート禁止区間と呼び、このアボート禁止区間内に当該プロセスの強制終了要求が入った場合は、アボート禁止区間が終了するまでプロセスの実行を継続し、アボート禁止解除が行われた時点で、当該プロセスを強制終了させる。
【００１９】
また、共有資源の使用時間が長い場合については、システム内にその処理を専門に行うプロセスを１個だけ用意し、更に、そのプロセスに要求を送るためのキューを用意する。そして、他の連続メディア処理を行うプロセスは、周期駆動を開始する前に専用プロセスに処理を依頼し、共有資源への処理が完了した後、周期駆動に入る。処理要求は先のキューに入れる。専用プロセス内では、共有資源に対する処理を行うプロセスがシステム内で１個であることが保証されるため、プリエンプト可能状態のまま共有資源に対する処理を行うことができ、周期駆動プロセスと並行して動作することができる。
【００２０】
これら二つのプロセス実行方法を組み合わせることで、複数のプロセスが並列に動作するマルチプロセシング環境において、他の周期駆動を行っているプロセスに影響を与えずに、共有資源を使用する複数のプロセスを実行させることが可能となる。
【００２１】
【発明の実施の形態】
以下、本発明の実施形態を詳細に説明する。
【００２２】
本発明に必要なモジュール群及びデータを図１に示す。

【００２３】
図１において、１０はプロセスを管理するためのテーブルであり、この中にアボート要求フラグ１１、アボート禁止フラグ１２、アボート禁止のネストをカウントするためのカウンタ１３を保持している。また、２０はプロセスを管理するプロセス管理部、２１はアボート禁止を行うためのインタフェース関数、２２はアボート禁止を解除にするためのインタフェース関数であり、２００は共有資源の処理を必要としているプロセスである。なお、プロセス２００内の共用資源を使用する関数４０は作業領域２３及び３７を保持している。更に、３０は特願平８－９７９９７に基づくプリエンプト制御部であり、スケジューラ３１、プリエンプト禁止関数３２、プリエンプト禁止解除関数３３、スケジューリング要求フラグ３４、プリエンプト禁止フラグ３５、プリエンプト禁止のネストをカウントするためのカウンタ３６を保持している。

【００２４】
本発明を実現するためのプロセス内のアボート禁止、プリエンプト禁止の制御の流れを図２に示す。共有資源に関わる処理を行う場合、その処理を行うプロセス２００内の関数４０は最初にアボート禁止関数２１を用いて自プロセスをアボート禁止にする（１０１）。そして、共有資源を使用する直前に自プロセスをプリエンプト禁止部３０内のプリエンプト禁止関数３２を用いてプリエンプト禁止にし（１０２）、共有資源を利用する処理に入る（１０３）。共有資源に対する処理を終了したら、直ちにプリエンプト禁止部３０内のプリエンプト禁止解除関数３３を用いてプリエンプト禁止を解除する（１０４）。ここで、プリエンプト禁止から禁止解除までの間（１０２～１０４）のプリエンプト禁止区間が設定できるのは、周期駆動性に影響を与えない時間、例えば周期駆動を管理する最小単位時間の１０％以下の時間で終了する共有資源への処理に限られる。次のプリエンプト禁止を宣言するまでの間はプリエンプト可能状態で、割り当て資源に対する初期化等、処理に長い時間を必要とする処理を行う（１０５）。この間は他のプロセスがスケジュールされ、プロセス２００の実行が一時中断されてもよい。そして再び共有資源への処理が必要になった場合は、再度プロセス２００は自プロセスをプリエンプト禁止状態にする（１０６）。そして、共有資源への処理（１０７）を行った後、プリエンプト禁止を解除する（１０８）。すべての処理が終了したら最後にアボート禁止解除関数２２を使ってアボート禁止を解除する（１０９）。

【００２５】
図２の例では、アボート禁止からアボート禁止解除までの間に２回のプリエンプト禁止区間が存在するが、実際に存在するプリエンプト禁止区間の数は３個以上でもよい。

【００２６】
図３は図２におけるアボート禁止の手続き１０１の内部処理を詳細に示したフローチャートである。図３の処理は図１に示したアボート禁止関数２１が行う。まず、アボート禁止関数２１は、プロセス２００を管理しているプロセス管理テーブル１０内のアボート禁止フラグ１２がＯＦＦの状態かどうかを調べる（１２０）。アボート禁止フラグ１２がＯＦＦであるならば、これをＯＮにする（１２１）。その後、カウンタ１３の値を１上げる（１２２）。図３の処理は、全体を不可分に行う。

【００２７】
図４は図２におけるプリエンプト禁止の手続き１０２及び１０６の内部処理を詳細に示したフローチャートである。図４の処理は図１に示したプリエンプト禁止関数３１が行う。まず、プリエンプト禁止関数３１は、プリエンプト制御部３０内のプリエンプト禁止フラグ３５がＯＦＦの状態かどうかを調べる（１３０）。アボート禁止フラグ３５がＯＦＦであるならば、これをＯＮにする（１３１）。その後、カウンタ３６の値を１上げる（１３２）。図４の処理は、全体を不可分に行う。

【００２８】
図５は図２におけるプリエンプト禁止解除の手続き１０４及び１０８の内部処理を詳細に示したフローチャートである。図５の処理は図１に示したプリエンプト禁止解除関数３２が行う。まず、プリエンプト禁止解除関数３２は、第１引数で渡される関数のネストの値

（後述）がカウンタ３６の値と一致してるか非かを調べる（１４０）。もし、一致していないければ異常処理を行う。一致している場合は、プリエンプト制御部３０内のカウンタ３６の値を１減らす（１４１）。その結果、カウンタ３６の値が正の値ならば、そのまま処理を終了する（１４２）。０又は負の値になったならば、プリエンプト禁止フラグをＯＦＦにする（１４３）。次に、プリエンプト制御部３０内のスケジューリング要求フラグ３４を調べる（１４４）。ＯＦＦならば処理を終了する。ＯＮならばスケジューラ３１を起動し（１４５）、プロセスのスケジューリング処理に入る。

【００２９】

図６は図２におけるアボート禁止解除の手続き１０９の内部処理を詳細に示したフローチャートである。図６の処理は図１に示したアボート禁止解除関数２２が行う。まずアボート禁止解除関数２２は、第１引数で渡される関数のネストの値（後述）がカウンタ１３の値と一致してるか非かを調べる（１５０）。もし、一致していないければ異常処理を行う。一致している場合は、プロセス２００を管理しているプロセス管理テーブル１０内のカウンタ１３の値を１減らす（１５１）。その結果、カウンタ１３の値が正の値ならば、そのまま処理を終了する（１５２）。０又は負の値になったならば、アボート禁止フラグをＯＦＦにする（１５３）。次に、プロセス管理テーブル１０内のアボート要求フラグ１１を調べる（１５４）。ＯＦＦならば処理を終了する。ＯＮならばプロセス管理部２０が起動し、プロセス２００のアボート処理に入る（１５５）。

【００３０】

また、図６において、アボート要求フラグ１１がＯＦＦの状態で処理が終了した場合、プロセス管理部２０はプリエンプト禁止部３０内のスケジューラ３１を起動し、プロセスの再スケジューリングを行う。

【００３１】

図１において、アボート禁止フラグ１２がＯＮになっている間、すなわち、アボート禁止関数２１が呼び出されてアボート禁止フラグ１２がＯＮにされてから、アボート禁止解除関数２２が呼び出されてアボート禁止フラグ１２がＯＦＦにされるまでの間がアボート禁止区間となる。アボート禁止フラグ１２はプロセス生成時にはＯＦＦになっており、アボート禁止関数２１が呼び出されることによって初めてＯＮになる。プロセス２００がアボート禁止関数２１を呼び出してアボート禁止フラグ１２をＯＮにした後、アボート禁止区間内、すなわち、アボート禁止フラグ１２がＯＮになっている間にプロセス２００の強制終了が発生した場合、プロセス管理部２０はアボート要求フラグ１１をＯＮにする処理だけを行い、プロセス２００の処理は続行させる。そして、プロセス２００がアボート禁止解除関数２２を呼び出してアボート禁止フラグをＯＦＦにした時に、プロセス２００の強制終了を行う。

【００３２】

また、図１において、プリエンプト禁止フラグ３５がＯＮになっている間、すなわち、プリエンプト禁止関数３２が呼び出されてプリエンプト禁止フラグ３５がＯＮにされてから、プリエンプト禁止解除関数３３が呼び出されてプリエンプト禁止フラグ３５がＯＦＦにされるまでの間がプリエンプト禁止区間となる。プリエンプト禁止フラグ３５はシステム起動時にはＯＦＦになっており、プリエンプト禁止関数３２が呼び出されることによって初めてＯＮになる。プロセス２００がプリエンプト禁止関数３２を呼び出してプリエンプト禁止フラグをＯＮにした後、プリエンプト禁止フラグ３５がＯＮになっている間に他のプロセスへのスケジューリング要求が発生した場合、スケジューラ３１はスケジューリング要求フラグ３４をＯＮにする処理だけを行い、他のプロセスのスケジューリングは行わず、プロセス２００の処理を続行させる。そして、プロセス２００がプリエンプト禁止解除関数３３を呼び出してプリエンプト禁止フラグ３５をＯＦＦにした時に、スケジューラ３１はスケジューリング要求フラグをＯＦＦにし、プロセス２００の実行を中断させ、他のプロセスをスケジュールする。

【００３３】

図２において、１０２～１０４及び１０６～１０８の間はプリエンプト禁止区間となり、

CPU使用権が他のプロセスへ移ることはないので、共有資源を使用するのは走行中のプロセスだけとなる。したがって、他のプロセスが共有資源を使用することはなく、資源の排他処理が実現でき、共有資源を使用する連続メディア処理のマルチプロセシングを行うことができる。また、１０１～１０９の間はアボート禁止区間となるため、強制終了によってプロセスが資源を獲得したまま消滅し、資源が永久に使用不能になることもない。

【００３４】

アボート禁止、アボート禁止解除は、それぞれ以下のインタフェースにより行う。

【００３５】

＜関数名＞

abort_disable(*level)

＜引数＞

level：本関数と関数abort_enableで形成される対のネストの深さが返される。

【００３６】

＜説明＞

本関数は、実行中のプロセスをアボート禁止状態に遷移させる。本関数を発行してアボート禁止状態に遷移してから関数abort_enalbeを発行してアボート可能状態に復帰するまでの区間で、本関数と関数abort_enableを対にして発行してもよい。すなわち、本関数と関数abort_enableの対はネスト可能である。ネストの内側で発行される本関数と関数abort_enableでは、アボート禁止状態とアボート可能状態との間の状態遷移は行わない。引数levelには、このネストの深さが返る。

【００３７】

＜関数名＞

abort_enable(level)

＜引数＞

level：本関数と対になる関数abort_disableが返したネスト・レベルを指定する。

【００３８】

＜説明＞

本関数は、実行中のプロセスをアボート可能状態に復帰させる。関数abort_disableを発行してアボート禁止状態に遷移してから本関数を発行してアボート可能状態に復帰するまでの区間で、関数abort_disableと本関数を対にして発行してもよい。すなわち、関数abort_disableと本関数の対はネスト可能である。ネストの内側で発行される関数abort_disableと本関数では、アボート禁止状態とアボート可能状態との間の状態遷移は行わない。引数levelには、このネストの深さ、すなわち対となる関数abort_disableの発行により得られたlevelの値を指定する。オペレーティング・システム側でもこの値を保持しており、引数で指定されたネストの深さと一致しない場合、エラーリターンする。

【００３９】

プロセス２００内の関数４０がアボート禁止関数２１を使用する際には、アボート禁止関数２１より返される現在のネストの深さを自関数内の作業領域２３に格納する。そして、アボート禁止解除関数２２を発行する際に、その作業領域に格納されている値を同関数の引数として指定する。アボート禁止解除関数では、この値と現在のネストの深さを比較し、両者が一致しない場合はエラーリターンする。この機能により、アボート禁止関数と対になるべきアボート禁止解除関数が存在しないというプログラムミスを容易に検出することができる。

【００４０】

また、プリエンプト禁止、プリエンプト禁止解除の処理は、特願平８－９７９９７において発明されているプリエンプト禁止、プリエンプト禁止解除のインタフェースを使用して行う。すなわち、次のインタフェースを使用する。

【００４１】

＜関数名＞

preempt_disable(*level)

＜引数＞
level：本関数と関数 preempt_enable で形成される対のネストの深さが返される。
【００４２】
＜説明＞
本関数は実行中タスクをプリエンプト禁止状態に遷移させる。本関数を発行しプリエンプト禁止状態に遷移してから、関数 preempt_enable を発行してプリエンプト可能状態に復帰するまでの区間で、本関数と関数preempt_enableを対にして発行しても良い。すなわち、本関数と関数 preempt_enable の対はネスト可能である。ネストの内側で発行される本関数と関数 preempt_enable は、プリエンプト禁止状態とプリエンプト可能状態との間の状態遷移は行なわない。level には、このネストの深さが返る。
【００４３】
＜関数名＞
preempt_enable(level)
＜引数＞
level：本関数と対となる関数 preempt_disable が返したネスト・レベルを指定する。
【００４４】
＜説明＞
本関数は実行中タスクをプリエンプト可能状態に復帰させる。関数 preempt_disable を発行しプリエンプト禁止状態に遷移してから、本関数を発行してプリエンプト可能状態に復帰するまでの区間で、関数 preempt_disable と本関数を対にして発行しても良い。すなわち、関数 preempt_disable と本関数の対はネスト可能である。ネストの内側で発行される関数 preempt_disable と本関数は、プリエンプト禁止状態とプリエンプト可能状態との間の状態遷移は行なわない。level には、このネストの深さ、すなわち対となる関数 preempt_enable の発行により得られた level の値を指定する。カーネルもこのネストの深さを保持しており、引数で指定されたネストの深さと一致しない場合には、エラーリターンする。
【００４５】
プロセス２００内の関数４０がプリエンプト禁止関数３１を使用する際には、プリエンプト禁止関数３１より返される現在のネストの深さを自関数内の作業領域３７に格納する。そして、プリエンプト禁止解除関数３２を発行する際に、その作業領域に格納されている値を同関数の引数として指定する。プリエンプト禁止解除関数では、この値と現在のネストの深さを比較し、両者が一致しない場合はエラーリターンする。この機能により、プリエンプト禁止関数と対になるべきプリエンプト禁止解除関数が存在しないというプログラムミスを容易に検出することができる。
【００４６】
次に、共有資源の使用時間が長い場合の、共有資源を使用するプロセスの実行方法の実施形態について説明する。
【００４７】
このプロセス実行方法の実施形態では、周期駆動を行うプロセスは、周期駆動開始前の周期駆動を行っていない状態の時に、次に説明するプロセス実行方法を用いて必要な資源の獲得等を行い、その処理が完了した後、自プロセスの周期駆動を開始する。周期駆動開始後は、次に説明するプロセス実行方法では資源の獲得は行わない。
【００４８】
本発明に必要なモジュール群を図７に示す。
【００４９】
図７において２１０～２１２は共有資源を使用したいプロセスであり、２２０は特定の共有資源に処理を行うことができる唯一のプロセスである。このプロセスを以下逐次化プロセスと呼ぶ。逐次化プロセス２２０はシステム内に常駐している。３００はプロセス２１０～２１２の処理要求を逐次化プロセス２２０に通知するための処理要求通知部であり、逐次化プロセス２２０に対してただ１個だけ存在し、逐次化プロセス２２０と同様常にシ

ステム内に常駐している。３１０～３１２はプロセス２２０からの処理完了通知をプロセス２１０～２１２に通知するための処理完了通知部であり、プロセス２１０～２１２の生成時、または処理要求の発生時に割り当てられる。処理要求通知部３００及び処理完了通知部３１０～３１２はそれぞれ内部に待ち行列４００、４１０～４１２を持っている。

【００５０】

次に、図７の処理を図８～１１のフローチャートを使って説明する。

【００５１】

図８は、共有資源に対する処理を必要とするプロセス２１０～２１２の処理を説明したフローチャートである。プロセス２１０～２１２はその共有資源に処理を行うことができるプロセス２２０に処理を要求するため、処理要求通知部３００に処理要求の通知を行う（１６０）。通知終了後、通知を行ったプロセス２１０～２１２は直ちに処理完了受理待ちになる（１６１）。処理完了通知部３１０～３１２から処理完了が通知されたら（１６１）、処理完了通知部３１０～３１２内の待ち行列４１０～４１２から処理終了通知を受理し（１６２）、共有資源に対する処理を終了する。

【００５２】

図９は、処理要求通知部３００の処理を説明したフローチャートである。処理要求通知部３００は常に処理要求通知の受信待ちになっている（１７１）。そして、処理要求の通知を受け取ったら（１７０）、受け付け順に処理要求を待ち行列４００に入れ（１７１）、処理要求が発生したことをプロセス２２０に通知する（１７２）。

【００５３】

図１０は、共有資源に処理を行うことができる逐次化プロセス２２０の処理を説明したフローチャートである。逐次化プロセス２２０も処理要求通知部３００と同様、常に処理要求通知の受信待ちになっている（１８０）。処理要求通知部３００から処理要求発生の通知を受けると（１８０）、処理要求通知部３００内の待ち行列４００から処理要求を受理し（１８１）、要求に応じた共有資源に対する処理を行う（１８２）。処理終了後、プロセス２２０は処理を要求したプロセス２１０～２１２の処理完了通知部３１０～３１２に処理完了及び処理結果の通知を行う（１８３）。逐次化プロセス２２０は、次の要求が通知されているかを調べ、通知されている場合は次の要求処理を行う。通知がない場合は次の要求受理待ちになる（１８０）。

【００５４】

図１１は、処理完了通知部３１０～３１２の処理を説明したフローチャートである。処理完了通知部３１０～３１２は、プロセス２１０～２１２が処理要求を発行したと同時に完了通知受信待ちになる（１９０）。そして、逐次化プロセス２２０から処理完了を受信したら（１９０）、受理した処理完了通知と処理結果を待ち行列４１０～４１２に入れ（１９１）、プロセス２１０～２１２に処理完了を通知する（１９２）。

【００５５】

図７の例では、共有資源の処理を要求するプロセスが３個であったが、２個以下または４個以上の場合でも処理は同じである。

【００５６】

図７における逐次化プロセス２２０は常にプリエンプト可能状態のまま動作する。

【００５７】

また、プロセス２１０～２１２は、上記の処理（図８）を周期駆動開始前、すなわち、周期駆動に関係のない状態の時に行い、上記処理の完了によって資源の獲得等が完了した後、周期駆動を開始させる。

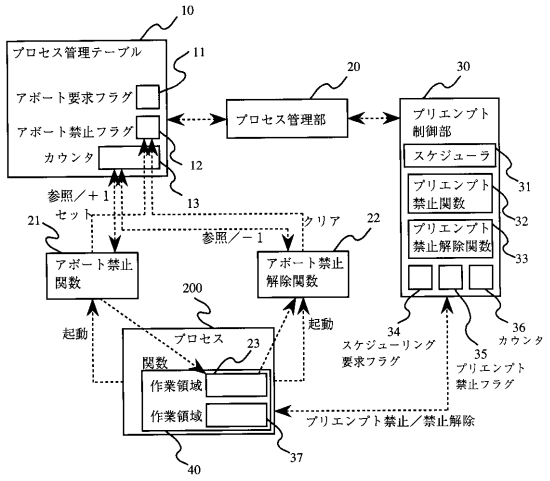
【００５８】

上記のような逐次化プロセスによるプロセス実行方法を用いると、共有資源に直接処理を行うプロセスが逐次化プロセスただ１個であるため、共有資源の排他制御を行う必要がない。また、共有資源に対する処理を専用の独立したプリエンプト可能なプロセスが行うため、他の連続メディア処理を行うプロセスの周期駆動性に影響を与えないプロセス実行が可能である。

【００５９】
【発明の効果】
以上述べたように、本発明によれば、共有資源を使用するマルチプロセシングの環境において、連続メディア処理等を行うプロセスの周期駆動性に影響を与えずに、プロセス間で共有資源を使用できる。
【図面の簡単な説明】
【図１】本発明に関わるモジュール群、及びデータの構成図である。
【図２】図１の実施例の処理手順を示すフローチャートである。
【図３】図１におけるアボート禁止関数２１の処理手順を示すフローチャートである。
【図４】図１におけるプリエンプト禁止関数３１の処理手順を示すフローチャートである。
【図５】図１におけるプリエンプト禁止解除関数３２の処理手順を示すフローチャートである。
【図６】図１におけるアボート禁止解除関数２２の処理手順を示すフローチャートである。
【図７】他の実施形態に関わるモジュール群の構成図である。
【図８】図７におけるプロセス２１０～２１２の処理手順を示すフローチャートである。
【図９】図７における処理要求通知部３００の処理手順を示すフローチャートである。
【図１０】図７における逐次化スレッド２２０の処理手順を示すフローチャートである。
【図１１】図７における処理完了通知部３１０～３１２の処理手順を示すフローチャートである。
【符号の説明】
１０……プロセス管理テーブル、１１……アボート要求フラグ、１２……アボート禁止フラグ、２０……プロセス管理部、２１……アボート禁止関数、２２……アボート禁止解除関数、３０……プリエンプト制御部、３１……スケジューラ、３２……プリエンプト禁止関数、３３……プリエンプト禁止解除関数、３４……スケジューリング要求フラグ、３５……プリエンプト禁止フラグ、２２０……逐次化プロセス

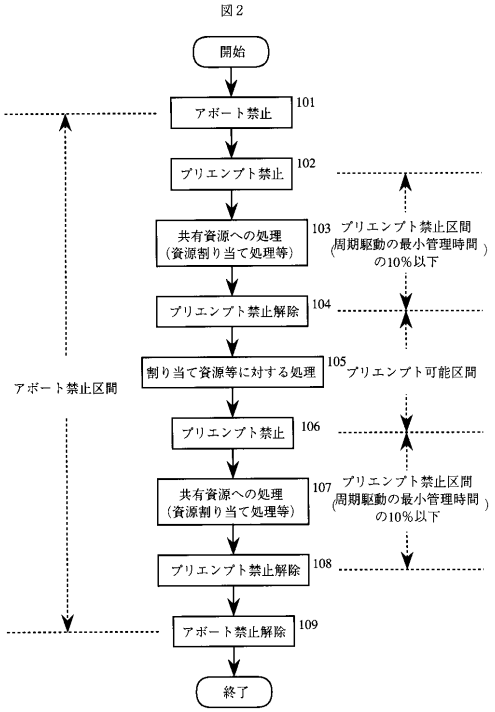
【図１】

図１



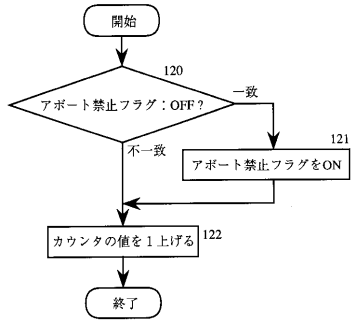
【図２】

図２



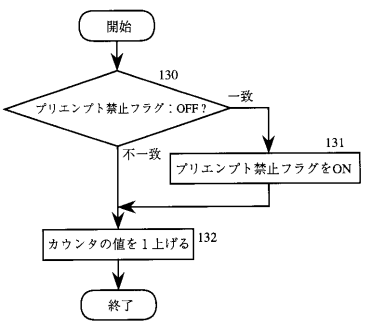
【図３】

図３



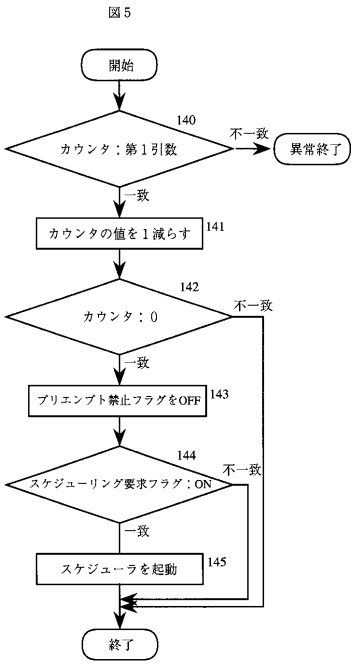
【図４】

図４



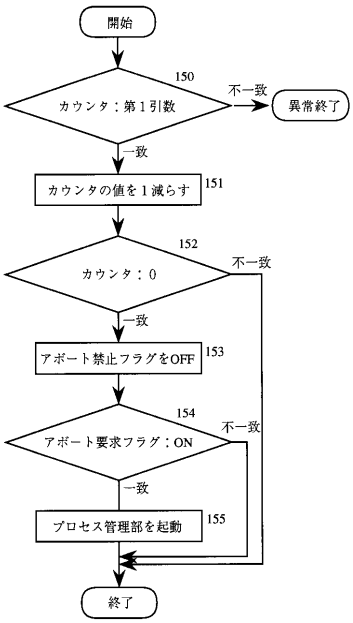
【図５】

図５



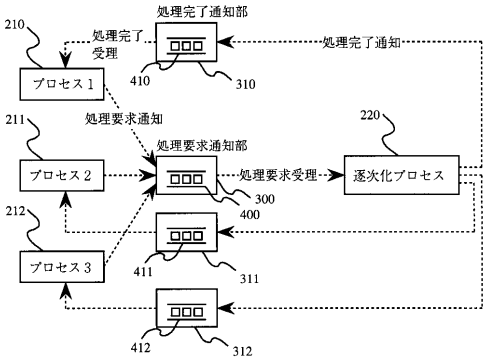
【図６】

図６



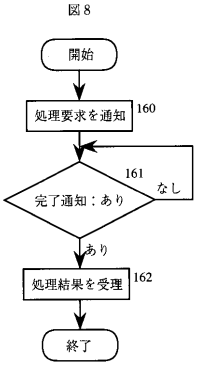
【図７】

図７



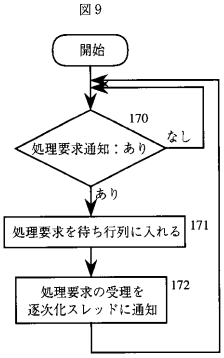
【図８】

図８



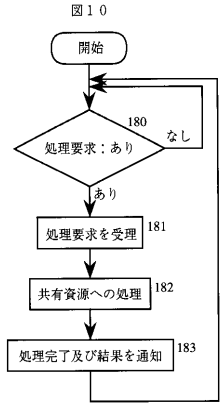
【図９】

図９



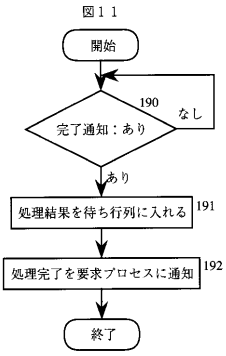
【図１０】

図１０



【図１１】

図１１



フロントページの続き

(72)発明者　竹内　理
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　中野　隆裕
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　芹沢　一
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

審査官　殿川　雅也

(56)参考文献　特開平１０－０６３５１５（ＪＰ，Ａ）
特開平０９－３１１７９４（ＪＰ，Ａ）
特開平０７－０１３７８２（ＪＰ，Ａ）
特開平０９－２８２１８２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.^{7}，ＤＢ名)
G06F　9/46　-　9/54