República Federativa do Brasil
Ministério do Desenvolvimento, Indústria
e do Comércio Exterior
Instituto Nacional da Propriedade Industrial

(11) **PI 0314853-0 B1**

(22) Data do Depósito: 11/08/2003

(45) Data de Concessão: 22/03/2016

(RPI 2359)

(54) Título: MÉTODO PARA MANIPULAÇÃO DE THREADS EM UM SISTEMA DE PROCESSAMENTO DE DADOS E SISTEMA DE PROCESSAMENTO DE DADOS

(51) Int.Cl.: G06F 9/46

(30) Prioridade Unionista: 19/09/2002 US 10/246,889

(73) Titular(es): INTERNATIONAL BUSINESS MACHINES CORPORATION

(72) Inventor(es): ALFREDO MENDOZA, JOEL HOWARD SCHOPP

Relatório Descritivo da Patente de Invenção para: **"MÉTODO PARA MANIPULAÇÃO DE THREADS EM UM SISTEMA DE PROCESSAMENTO DE DADOS E SISTEMA DE PROCESSAMENTO DE DADOS".**

Antecedentes da Invenção

Campo Técnico

[0001] A presente invenção se refere, de um modo geral, a um sistema de processamento de dados aperfeiçoado e, em particular, a um método e a um aparelho para gerenciamento de threads em um sistema de processamento de dados.

Descrição da Técnica Relacionada

[0002] Uma thread é uma unidade básica de utilização da unidade central de processamento (CPU) Uma thread, usualmente, tem um contador de programa, um conjunto de registradores e um espaço na pilha. Uma thread compartilha com outras threads sua seção de código, seção de dados e recursos operacionais do sistema, tais como arquivos e sinais. Esses componentes também são conhecidos como uma "tarefa". Alguns sistemas implementam threads de usuário em bibliotecas de nível de usuário, em lugar de utilizar as chamadas do sistema, de modo que a comutação de thread não precisar chamar o sistema operacional e causar uma interrupção no kernel.

[0003] As threads operam, em muitos aspectos, da mesma maneira que os processos. Uma thread pode estar em um de diversos estados: pronta, bloqueada, em espera, processando ou terminada. As threads de usuário em um espaço de usuário são processadas por threads de kernel em um kernel. Uma thread de kernel também é referida como um "processador virtual". Em alguns casos, um modelo de um para um é usado, em que cada thread de usuário tem uma thread de kernel correspondente. Em outros casos, um modelo de M:N é usado, em que muitas threads de usuário são processadas em umas poucas threads de kernel para promover um desempenho acentuado. Com esse modelo, situações ocorrem, tais como bloqueio em um mutex, em que uma thread de kernel não é mais necessária para processar uma thread de usuário. Um mutex é um mecanismo de bloqueio envolvendo o uso de um flag de programação usado para bloquear e liberar um objeto. Quando são adquiridos dados que não podem ser compartilhados ou é iniciado o processamento que não pode ser realizado simultaneamente em qualquer parte no sistema, o mutex é ajustado para "travar", o que bloqueia outras tentativas para usá-lo. O mutex é ajustado para "destravar", quando os dados não são mais necessários ou a rotina está acabada. Se nenhuma outra thread de usuário for processável no momento,

essa thread de kernel se separará daquela thread de usuário particular e entrará em um estado de espera.

[0004] A separação de uma thread de kernel para entrar em um estado de espera resulta em um número de ações ocorrendo. Uma ação empreendida é que a thread de kernel comuta de uma pilha de usuário para sua própria pilha menor. Adicionalmente, a thread de kernel habilita o mascaramento de sinal para bloquear a maioria dos sinais. Quando a thread de kernel é necessária mais uma vez, essa thread comutará para a pilha da thread de usuário e habilitará diversos atributos específicos de thread, tais como máscaras de sinais.

[0005] A presente invenção reconhece que esse mecanismo de separação e subsequente re-conexao usados correntemente envolve um grande overheard de desempenho. Cada separação ou re-conexão requer uma chamada de sistema para copiar dados do espaço do usuário para o espaço do kernel ou para o espaço do kernel a partir do espaço do usuário. Adicionalmente, diversos bloqueios são usados na biblioteca do kernel e no kernel, resultando em contenção de bloqueio possivelmente crescente. Esse tipo de separação também envolve problemas potenciais de manipulação de sinais. Especificamente, uma pequena janela está presente antes que a thread de kernel bloqueie os sinais onde, a

thread de kernel poderia receber um sinal enquanto executando em sua pilha de kernel pequena. A presente invenção também reconhece que um manipulador de sinais, que funciona bem em uma pilha de thread de usuário maior, pode estourar a pilha de thread de kernel menor, corrompendo a memória e/ ou fazendo com que o aplicativo passe por uma descarga de memória.

[0006] Esses dois problemas de baixo desempenho e estouro de pilha são problemas distintos, mas tem uma causa de origem similar. Essa causa é a separação das threads de kernel "inativas". Portanto, seria vantajoso ter um método, aparelho e instruções de computador aperfeiçoados para manipulação das threads de kernel inativas de uma maneira que reduza o baixo desempenho e evite o estouro da pilha.

Sumario da Invenção

[0007] A presente invenção proporciona um método, um aparelho e instruções de computador para gerenciamento de threads. Uma thread de kernel associada com uma thread de usuário é detectada como sendo desnecessária pela thread de usuário. A thread de kernel é parcialmente separada em que os dados para a thread não mudam pilhas em resposta à thread de kernel sendo desnecessária.

[0008] De preferência, a thread de kernel é colocada em uma lista parcialmente separada e em um estado não processável simultaneamente corna thread de usuário.

[0009] De preferência, quando a thread de usuário desperta da espera, a thread de kernel é removida da lista parcialmente separada de modo que a re-conexao da thread de usuário é desnecessária.

[0010] De preferência, as threads executam em um sistema operacional AIX.

[0011] De preferência, os dados para a thread de kernel permanecem em uma pilha de usuário, sem requerer a cópia dos dados para uma pilha de kernel para parcial separação da thread de kernel.

[0012] De preferência, a detecção de que uma thread de kernel é desnecessária e a separação parcial de thread de kernel são realizadas usando uma biblioteca.

[0013] De preferência, quando a thread de usuário sai do estado de espera, é determinado se uma segunda thread de kernel é separada em lugar de parcialmente separada e, se a segunda thread de kernel for separada, ela é anexada à thread de usuário.

[0014] De preferência, o estado não processável é um estado de espera ou de um estado de repouso.

[0015] De acordo com outro aspecto, a presente invenção proporciona um sistema de processamento de dados para gerenciamento de threads, o sistema de processamento de dados compreendendo: um sistema de barramento; uma unidade de comunicação conectada ao sistema de barramento; uma memória conectada ao sistema de barramento, em que a memória inclui um conjunto de instruções; e uma unidade de processamento conectada ao sistema de barramento, em que a unidade de processamento executa o conjunto de instruções para detectar uma thread de kernel associada com uma thread de usuário como sendo desnecessária pela thread de usuário; e a separação parcial da thread de kernel, em que os dados para a thread não mudam pilhas em resposta a thread de kernel sendo desnecessária.

Breve Descrição dos Desenhos

[0016] A presente invenção será agora descrita, à guisa de exemplo apenas, com referência a uma concretização preferida, como ilustrado nos desenhos anexos, em que:

[0017] A figura 1 é uma representação pictorial de um sistema de processamento de dados em que a presente invenção pode ser implementada de acordo com uma concretização preferida da presente invenção.

[0018] A figura 2 é um diagrama em blocos de um sistema de processamento de dados em que a presente invenção pode ser implementada.

[0019] A figura 3 e um diagrama ilustrando componentes usados na manipulação de threads de kernel.

[0020] As figuras 4A - 4C são ilustrações de estruturas de dados usadas em uma lista parcialmente separada.

[0021] A figura 5 é um fluxograma de um método conhecido usado para manipulação de chamadas de bloqueio.

[0022] A figura 6 é um fluxograma de um método usado para gerenciamento de threads inativas.

[0023] A figura 7 é um fluxograma de um método conhecido usado para ativar uma thread de kernel.

[0024] A figura 8 é um fluxograma de um método usado para manipulação de uma thread de kernel.

Descrição Detalhada da Concretização Preferida

[0025] Com referência agora às figuras e em particular com referência à figura 1, uma representação pictorial de um sistema de processamento de dados, em que a presente invenção pode ser implementada, é representado de acordo com uma concretização preferida da presente invenção. Um computador 100 é representado, o qual inclui uma unidade de sistema 102, um terminal de exibição de vídeo 104, um

teclado 106, dispositivos de armazenamento 108, que podem incluir discos flexíveis e outros tipos de mídias de armazenamento permanentes e removíveis e mouse 110. Dispositivos de entrada adicionais podem ser incluídos com computador pessoal 100, tal como, por exemplo, um joystick, uma *touchpad*, uma tela de toque, um *trackball*, um microfone e semelhantes. O computador 100 pode ser implementado usando qualquer computador adequado, tal como um computador IBM eServer ou um computador IntelliStation, que são produtos da International Business Machines Corporation, localizada em Armonk, New York. Embora a representação apresentada mostre um computador, outras concretizações da presente invenção podem ser implementadas em outros tipos de sistemas de processamento de dados, tais como um computador de rede. O computador 100 também inclui, de preferência, uma interface gráfica de usuário (GUI), que pode ser implementada por meio de software de sistemas residente em meio legível em computador em operação dentro do computador 100.

[0026] Com referência agora à figura 2, um diagrama em blocos de um sistema de processamento de dados é mostrado em que a presente invenção pode ser implementada. O sistema de processamento de dados 200 é um exemplo de um computador, tal como o computador 100 na figura 1, em que código ou instruções implementando os processos da presente invenção

podem ser localizadas. O sistema de processamento de dados 200 emprega uma arquitetura de barramento local de interligação de componentes periféricos (PCI). Embora o exemplo representado empregue um barramento PCI, outras arquiteturas de barramento, tais como Porta Gráfica Acelerada (AGP) e Arquitetura Padrão para a Industria (ISA), podem ser usadas. O processador 202 e a memória principal 204 são conectados ao barramento local PCI 206 através da ponte PCI 208. A ponte PCI 208 também pode incluir um controlador de memória integrado e memória cache para o processador 202. No exemplo representado, o adaptador de rede de área local (LAN) 210, o adaptador de barramento principal de pequena interface de sistema de computador SCSI 212 e a interface de barramento de expansão 214 são conectados ao barramento local PCI 206 por conexão direta de componentes. Em contraste, o adaptador de áudio 216, adaptador gráfico 218 e adaptador de áudio/ vídeo 219 são conectados ao barramento local PCI 206 por meio de placas de adição inseridas em *slots* de expansão. A interface de barramento de expansão 214 proporciona uma conexão para um adaptador de teclado e mouse 220, modem 222 e memória adicional 224. O adaptador de barramento principal SCSI 212 proporciona uma conexão para mecanismo de disco rígido 226, mecanismo de fita 228 e mecanismo de CD-ROM 230.

Implementações típicas de barramento local PCI suportarão três ou quatro *slots* de expansão ou conectores de introdução de adição.

[0027] Um sistema operacional roda no processador 202 e é usado para coordenar e proporcionar controle de vários componentes dentro do sistema de processamento 200 na figura 2. O sistema operacional pode ser um sistema operacional comercialmente disponível, tal como um *Advanced Interactive eXecutive* (AIX) ou Windows XP. AIX é uma versão do UNIX e está disponível a partir da International Business Machines Corporation. O Windows XP está disponível a partir da Microsoft Corporation. As instruções para o sistema operacional e aplicações ou programas estão localizadas nos dispositivos de armazenamento, tais como um mecanismo de disco rígido 226, e podem estar localizadas na memória principal 204 para execução pelo processador 202.

[0028] Aqueles de habilidade comum na técnica compreenderam que o hardware na figura 2 pode variar, dependendo da implementação. Outros hardware internos ou dispositivos periféricos, tais como memória flash somente de leitura (ROM), memória não volátil equivalente ou·mecanismos de discos óticos e semelhantes, podem ser usados em adição ou em lugar do hardware representado na figura 2. Também, os

processos da presente invenção podem ser aplicados a um sistema de processamento de dados de multiprocessador.

[0029] Por exemplo, o sistema de processamento de dados 200, se opcionalmente configurado como um computador em rede, pode não incluir um adaptador de barramento principal SCSI 212, mecanismo de disco rígido 226, mecanismo de fita 228 e o mecanismo de CD-ROM 230. Naquele caso, o computador, a ser chamado, adequadamente, de um computador cliente, inclui algum tipo de interface de comunicações de rede, tal como um adaptador LAN 210, um modem 222 ou semelhante. Como outro exemplo, o sistema de processamento de dados 200 pode ser um sistema autônomo configurado para ser inicializável, sem contar com algum tipo de interface de comunicações de rede, quer ou não o sistema de processamento de dados 200 compreenda algum tipo de interface de comunicações de rede. Como um outro exemplo, o sistema de processamento de dados 200 pode ser um assistente pessoal digital (PDA), que é configurado com Rom e/ ou ROM flash para proporcionar memória não volátil para armazenar arquivos de sistemas operacionais e/ ou dados gerados pelo usuário.

[0030] O exemplo representado na figura 2 e os exemplos descritos acima não implicam em limitações de arquitetura. Por exemplo, o sistema de processamento de dados

200 também pode ser um computador notebook ou um computador portátil além de tomar a forma de um PDA. O sistema de processamento de dados 200 também pode ser um quiosque ou um meio da Web.

[0031] Os processos da presente invenção são realizados pelo processador 202 usando instruções implementadas por computador, que podem estar localizadas em uma memória, tal como, por exemplo, memória principal 204, memória 224 ou em um ou mais dispositivos periféricos 226 - 230.

[0032] Voltando agora à figura 3, um diagrama ilustrando componentes usados na manipulação de threads de kernel é representado de acordo com uma concretização preferida da presente invenção. Neste exemplo, threads de usuário 300, 302, 304, 306 e 308 estão localizadas no espaço de kernel 310, enquanto as threads de Kernel 312, 314, e 316 estão localizadas no espaço de Kernel 318. Essas threads neste exemplo seguem o modelo M:N, em que muitas threads de usuário são processadas em umas poucas threads de kernel para aumentar o desempenho.

[0033] Atualmente, a thread de usuário 300 está sendo processada pela thread de kernel 312, a thread de usuário 304 está sendo processada pela thread de kernel 314 e a thread de usuário 306 está sendo processada pela thread de

kernel 316. As operações realizadas para essas threads de usuário estão localizadas nas pilhas de thread de usuário 320. Cada thread de usuário está associada com uma pilha de thread de usuário. As threads de kernel 312, 314 e 316 têm dados localizados nas pilhas de thread de usuário 320. A pilha particular nas pilhas de thread de usuário 320 é uma pilha associada com a thread de usuário que está sendo processada pela thread de kernel.

[0034] Normalmente, se uma thread de kernel, tal como a thread de kernel 312, não é mais necessaria para processar uma thread de usuário, tal como a thread de usuário 300, a thread de kernel 312 se separará e introduzirá um estado de espera junto com a thread de usuário 300. As threads de kernel em um estado de espera, norma1mente, são colocadas em lista separada 322, que e gerenciada pela biblioteca de Pthread 324. A biblioteca Pthread 324 é uma biblioteca carregável dinamicamente, que é usada em AIX. Com uma separação da thread de usuário 300, a informação para thread de kernel 312, que representa o indicador de pilha corrente que aponta para a pilha de thread de usuário em pilhas de thread de usuário 320, é modificada para apontar para a área reservada para sua pilha de thread de kernel nas pilhas de thread de kernel 326. Quando a thread de usuário 300 sai de um estado de espera, a thread de kernel 312 pode ser removida

da lista separada 322 e re-anexada à thread de usuário 300. Alternativamente, se a thread de kernel 312 estiver indisponível, outra thread de kernel disponível na lista separada 322 pode ser anexada à thread de usuário 300.

[0035] De acordo com uma concretização preferida da presente invenção, quando a thread de usuário 300 introduz um estado em que a thread de kernel 312 é desnecessária, tal como um estado de espera, a thread de kernel 312 é colocada na 1ista parcialmente separada 328, em lugar de na lista separada 322, pela biblioteca de *Pthread* 324. Com a separação parcial, a thread de kernel 312 não muda sua pi1ha ou máscara de sinal. Ao invés disso, a thread de kernel 312 é colocada na lista parcialmente separada 328 e fica em espera simultaneamente com a thread de usuário 300. Nesse estado de espera simultâneo, a thread de kernel 312 mantém a informação tal como a pilha e a máscara de sinal da thread de usuário pela qual está esperando, simultaneamente. Essa informação também é referida como "atributos específicos de thread de usuário". Com essa espera simultânea, a thread de kernel 312 se identifica como estando disponível para processar outras threads de usuário, mas preferirá uma thread na lista separada 322 a ser usada primeiro. A preferência é dada ao processamento da thread de usuário com que a thread de kernel 312 está associada, a thread de usuário 300. A lista

parcialmente separada 328 pode ser implementada de maneiras diferentes, dependendo da implementação particular. Neste exemplo, essa lista é implementada como um kernel ligado de estruturas de threads.

[0036] Se a thread de usuário 300 sai de um estado de espera, a thread de kernel 312 se removerá da lista parcialmente separada 328 e continuará processando a thread de usuário 300. Esse mecanismo proporciona um curso em que a latência menor está envolvida na ativação de uma thread na biblioteca de *Pthread* 324. Esse mecanismo é útil quando da ativação de um mutex contestado, uma variável de condição ou um sinal porque ações realizadas em seguida a esses eventos, frequentemente, precisam ser completadas, antes que o resto do programa possa progredir.

[0037] Se uma thread de kernel na lista parcialmente separada 328 é necessária para processar outra thread de usuário posteriormente, uma penalidade de desempenho para a separação restante pode ser evitada através da separação do estado da thread de usuário corrente para o estado da nova thread de usuário, sem separação da thread de kernel. Por exemplo, se a thread de kernel 312 está na lista parcialmente separada 328 e outra thread de usuário, tal como a thread de usuário 302, requer a thread de kernel 312 para processar aquela thread, a pilha associada com a thread de usuário 300

pode ser anexada à thread de usuário 302, através da mudança das propriedades dentro da pilha para a thread de usuário 300 para correspondência com aquelas da thread de usuário 302. Por exemplo, o indicador de pilha e a máscara de sinal podem ser mudados daquelas para a thread de usuário 300 para a thread de usuário 302.

[0038] Como um resultado, uma thread de kernel só pode se separar completamente quando uma thread de usuário associada com a thread de kernel sai ou termina. Consequentemente, o baixo desempenho associado com as separações normais é evitado na maioria dos casos.

[0039] Voltando a seguir às figuras 4A - 4C, ilustrações de estruturas de dados usadas em uma lista parcialmente separada são representadas de acordo com uma concretização preferida da presente invenção. Essas estruturas de dados podem ser usadas para implementar uma lista parcialmente separada, tal como a lista parcialmente separada 328, na figura 3.

[0040] Na figura 4A, uma lista encadeada e usada para indicar as threads de kernel diferentes. Por exemplo, as entradas 400 e 402 são usadas para indicar as threads de kernel diferentes. O marcador de cabeça de lista 406 identifica o começo da lista encadeada. A entrada 400 contém o indicador anterior 408 e o indicador seguinte 410. Esses

ponteiros são usados para indicar uma entrada anterior e a seguinte dentro da lista encadeada. Ainda, a entrada 400 inclui o ponteiro 416 e a entrada 402 inclui o ponteiro 418, com esses indicadores apontando para threads de kernel, tais como as threads de kernel 420 e 422.

[0041] A seguir, na figura 4B, a informação usada na estrutura da lista é incorporada nas estruturas de threads. Neste exemplo, o marcador de cabeça da lista 430 aponta para o começo ou primeira thread, a thread de kernel 432. A thread de kernel 432 contém o indicador seguinte 434 que aponta para a thread de kernel 436. A thread de kernel 432 também inclui o indicador anterior 438, que aponta para algum kernel anterior na lista. A thread de kernel 436 contém o ponteiro anterior 440, que aponta de volta para a thread de kernel 432. O ponteiro seguinte 442 na thread de kernel 436 aponta para a thread de kernel seguinte na lista. Este exemplo é a estrutura de lista preferida nos exemplos ilustrados.

[0042] Na figura 4C, o arranjo 450 é usado para apontar para as threads de kernel diferentes. O marcador de cabeça da lista 452 aponta para o começo da fila 450, que contém os ponteiros 454, 456 e 458. Esses indicadores apontam para as threads de kernel 460, 462 e 464, respectivamente. Esses exemplos são proporcionados apenas como ilustrações com relação a como uma lista parcialmente separada pode ser

implementada. Outros tipos de estruturas, tais como uma árvore, podem ser usadas, dependendo da implementação particular.

[0043] Com referência agora à figura 5, um fluxograma de um método conhecido usado para manipulação de chamadas de bloqueio é representado. O método ilustrado na figura 5 pode ser implementado em uma biblioteca, tal como a biblioteca de *Pthread* 324 na figura 3. Uma chamada de bloqueio é qualquer chamada que pode fazer com que uma thread ao nível do usuário mude de um estado de processamento·ou processável para outro estado que é um estado de repouso ou de espera.

[0044] O método começa pela detecção de uma chamada potencialmente de bloqueio (etapa 500). O método separa a thread de kernel da thread de usuário (etapa 502). A etapa 502 requer o baixo desempenho de copiar informações da pilha de thread de usuário para uma pilha de thread de kernel, bem como outras operações, tais como mudança de mascaramento de sinais para sinais de bloqueio. A seguir, o método procura por uma nova thread de usuário para processamento (etapa 504). Se a thread processável for encontrada, a thread de kernel se anexa à nova thread de usuário (etapa 506). A etapa 506 envolve baixo desempenho, tal como cópia de dados de uma pilha de thread de kernel para a pilha de thread de usuário, bem como estabelecimento de uma máscara de sinal. A nova

thread de usuário é processada (etapa 508) e o método termina em seguida.

[0045] Com referência mais uma vez à etapa 504, se não houver threads processáveis, a thread de kernel é colocada em uma lista separada e vai para um estado de espera (etapa 510). Em seguida, o método aguarda para detectar uma thread de usuário que se torna processável (etapa 512). Em seguida, a thread de kernel é anexada à thread de usuário que é separada como sendo processável (etapa 514). A thread de usuário é, então, processada pela thread de kernel (etapa 514). A thread de usuário é, então, processada pela thread de kernel (etapa 516) e o método termina em seguida.

[0046] Voltando agora à figura 6, um fluxograma de um método usado para gerenciamento de threads inativas é representado de acordo com uma concretização preferida da presente invenção. O método ilustrado na figura 6 pode ser implementado em uma biblioteca, tal como a biblioteca de *Pthread* 324 na figura 3.

[0047] O método começa pela detecção de uma chamada potencialmente de bloqueio (etapa 600). Essa chamada potencialmente de bloqueio é uma que colocada à thread ao nível do usuário em um estado de repouso ou de espera. Uma thread que está em espera ou em repouso é referida como uma thread "não processável". O método, então, procura por uma

nova thread de usuário para processar (etapa 602). Se uma thread de usuário processável for encontrada, o método comuta a thread de kernel para a nova thread de usuário (etapa 604). A comutação na etapa 604 pode ser realizada através do uso de um distribuidor ao nível do usuário. Dependendo da implementação particular, essa comutação pode ou não requerer a separação e re-conexao do thread de kernel. A thread de usuário é processada pela thread de kernel (etapa 606) com o método terminando em seguida.

[0048] Voltando mais uma vez a etapa 602, se nenhuma thread de usuário processável for encontrada, a thread de kernel é colocada em uma espera parcialmente separada (etapa 608). Nesse tipo de espera, a thread de kernel é colocada em uma lista parcialmente separada e, então, entra em um estado de espera. A thread de usuário associada com a thread de kernel é mantida ou relacionada na lista parcialmente separada com a thread de kernel. Essa associação é feita para indicar uma preferência para usar aquela thread de kernel particular para processar a thread de usuário, em lugar de processar outra thread de usuário, a menos que necessário.

[0049] Em seguida, um evento é detectado e processado (etapa 610). Se o evento for uma thread de usuário associada com a thread de kernel tornando-se processável, aquela thread

de usuário é processada pela thread de kernel (etapa 612) e o método termina em seguida. Com referência mais uma vez à etapa 610 se a thread de usuário associada com a thread de kernel sai ou, de outro modo, não tem necessidade da thread de kernel, a thread de kernel é colocada em uma lista separada e espera (etapa 614), com o método terminando em seguida. Mais especificamente, a etapa 614 introduz uma série de etapas ilustradas na figura 5, começando na etapa 510.

[0050] Voltando à etapa 610, se o evento for qualquer outro evento, o método procura uma thread de usuário para processar (etapa 616). Se nenhuma thread processável for encontrada, o método retorna para a etapa 608. Caso contrário, se outra thread de usuário se torna processável, o método comuta a thread de kernel para a thread de usuário que se tornou processável (etapa 618). Essa etapa pode incluir duas operações, uma separação e uma conexão. Alternativamente, uma única operação, em que a thread de kernel é comutada da thread de usuário corrente para a nova thread de usuário pode ser empregada. A thread de usuário é, então, processada pela thread de kernel (etapa 620) com o método terminando em seguida.

[0051] Com referência agora à figura 7, um fluxograma de um método conhecido usado para ativar uma thread de kernel é representado. O método ilustrado na figura 7 pode ser

implementado em uma biblioteca, tal como a biblioteca de *Pthread* 324 na figura 3. O método ilustrado nesta figura permite a uma thread aguardar ou esperar até que uma thread de usuário seja detectada como se tornando processável.

[0052] O método começa pela verificação de uma lista separada para threads de kernel (etapa 700). Se a lista separada não está vazia, uma thread de kernel é removida da lista separada (etapa 702). A thread de kernel é ativada (etapa 704) com o método terminando em seguida. Com referência mais uma vez à etapa 700, se a lista estiver vazia, o método termina.

[0053] Voltando agora à figura 8, um fluxograma de um método usado para manipulação de uma thread de kernel é representado de acordo com uma concretização preferida da presente invenção. O método ilustrado na figura 8 pode ser implementado em uma biblioteca, tal como uma biblioteca de *Pthread* 324 na figura 3. Esse método é iniciado em uma thread de kernel que faz outra thread de kernel processável.

[0054] O método começa pela verificação de uma lista separada para uma thread de kernel (etapa 800). Se a lista separada estiver vazia, o método verifica uma lista parcialmente separada para uma thread de kernel (etapa 802). Se a lista parcialmente separada estiver vazia, o método termina. Caso contrário, se a lista parcialmente separada

não estiver vazia, uma thread de kernel é removida da lista semi-separada (etapa 804). Essa thread de kernel, removida da lista parcialmente separada, é ativada (etapa 806) com o método terminando em seguida. Naquele ponto, a thread de kernel é processada.

[0055] Com referência mais uma vez à etapa 800, se a lista separada não está vazia, uma thread de kernel é removida da lista separada (etapa 808). A thread de kernel é ativada (etapa 810) com o método terminando em seguida. O mecanismo da presente invenção ainda emprega uma lista separada porque a separação, em lugar da parcial separação, pode ser necessária em algumas situações. Essa lista pode ser necessária para as threads de usuário existentes. Nesse caso, a seleção de uma thread de kernel da lista separada é preferida em relação à seleção de uma thread de kernel de uma lista parcialmente separada, quando da ativação das threads de usuário.

[0056] Desse modo, a presente invenção proporciona um método, um aparelho e instruções de computador aperfeiçoados para manipulação das threads de kernel inativas. O mecanismo da presente invenção evita o baixo desempenho correntemente envolvido com a separação e subsequente re-conexão de threads de kernel. Ainda, o mecanismo da presente invenção também evita o estouro de

pilha, que pode ocorrer. Essas vantagens são proporcionadas através do uso de um método de separação parcial. Esse método envolve a colocação da thread de kernel em uma lista parcialmente separada, e, então, colocar a thread de kernel em um estado de espera. Aquelas etapas ocorrem sem demandar as etapas que são requeridas, normalmente, na separação de uma thread de kernel de uma thread de usuário.

[0057] É importante notar que, embora a presente invenção tenha sido descrita no contexto de um sistema de processamento de dados funcionando completamente, aqueles de habilidade comum na técnica apreciarão que os processos da presente invenção são capazes de serem distribuídos na forma de um meio de instruções legível em computador e uma variedade de formas e que a presente invenção se aplica igualmente, independentemente do tipo particular de meios portadores de sinais realmente usados para realizar a distribuição. Exemplos de meios legíveis em computador incluem meios do tipo gravável, tais como disco flexível, um mecanismo de disco rígido, uma RAM, CD-ROMs, DVD-ROMs e meios do tipo transmissão, tais como ligações de comunicações digitais e analógicas, ligações de comunicações com fio e sem fio, usando formas de transmissão tais como, por exemplo, radiofrequência e transmissões de sinais de luz. Os meios legíveis em computador podem tomar a forma de formatos

codificados, que são decodificados para uso real em um sistema de processamento de dados particular.

REIVINDICACÕES

1. Método para manipulação de threads em um sistema de processamento de dados, o método **caracterizado pelo** fato de compreender:

detecção de uma thread de kernel (312, 420, 432) associada com uma thread de usuário (300) como sendo desnecessária pela thread de usuário; e

em resposta à thread de kernel (312, 420, 432) sendo desnecessária, separação parcial da thread de kernel (312, 420, 432), em que os dados para a thread não mudam pilhas.

2. Método, de acordo com a reivindicação 1, **caracterizado pelo** fato de ainda compreender:

colocação da thread de kernel (312, 420, 432) em uma lista parcialmente separada (328); e

colocação da thread de kernel (312, 420, 432) em um estado não processável simultâneo com a thread de usuário (300).

3. Método, de acordo com a reivindicação 2, **caracterizado pelo** fato de ainda compreender, em resposta à ativação da thread de usuário da espera, remoção da thread de kernel (312, 420, 432) da lista parcialmente separada (328), em que a re-conexão à thread de usuário (302) é desnecessária.

4. Método, de acordo com a reivindicação 2, **caracterizado pelo** fato de ainda compreender:

em resposta à ativação da thread de usuário (304) do estado de espera, determinação de se uma segunda thread de kernel (314, 422, 436) está separada (322) em lugar de parcialmente separada (328); e

em resposta à segunda thread de kernel (314, 422, 436) estar separada (322), anexação da segunda thread de kernel (314, 422, 436) à thread de usuário (304).

5. Método, de acordo com a reivindicação 2, **caracterizado pelo** fato de o estado não processável ser um de um estado de repouso ou estado de espera.

6. Sistema de processamento de dados (200) para gerenciamento de threads, o sistema de processamento de dados **caracterizado pelo** fato de compreender:

meios de detecção de thread de kernel (312, 420, 432) associada com uma thread de usuário (300) como sendo desnecessária pela thread de usuário; e

meios de separação parcial, em resposta à thread de kernel (312, 420, 432) ser desnecessária para a separação parcial da thread de kernel (312, 420, 432), em que dados para a thread não mudam pilhas.

7. Sistema de processamento de dados, de acordo com a reivindicação 6, **caracterizado pelo** fato de ainda compreender:

primeiro meio de colocação para colocar a thread de kernel (312, 420, 432) em uma lista parcialmente separada (328); e

segundo meio de colocação para colocação da thread de kernel (312, 420, 432) em um estado não processavel simultâneo com a thread de usuário (300).

8. Sistema de processamento de dados, de acordo com a reivindicação 7, **caracterizado pelo** fato de ainda compreender meios de remoção, em resposta à ativação da thread de usuário da espera, para remoção da thread de kernel da lista parcialmente separada (328), em que a re-conexao é desnecessária.

9. Sistema de processamento de dados, de acordo com a reivindicação 7, **caracterizado pelo** fato de ainda compreender:

meios de determinação, em resposta à ativação de thread de usuário (304) do estado de espera, para determinação se uma segunda thread de kernel (314, 422, 436) está separada (322) em lugar de parcialmente separada (328); e

meios de anexação, em resposta à segunda thread de kernel ser separada (322), para anexação da segunda thread de kernel (314, 422, 436) à thread de usuário (304).

10. Sistema de processamento de dados, de acordo com a reivindicação 7, **caracterizado pelo** fato de o estado não processável ser um de um estado de repouso ou um estado de espera.

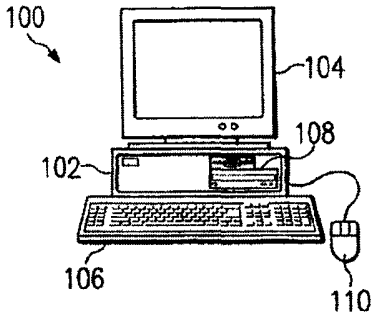


FIGURA 1

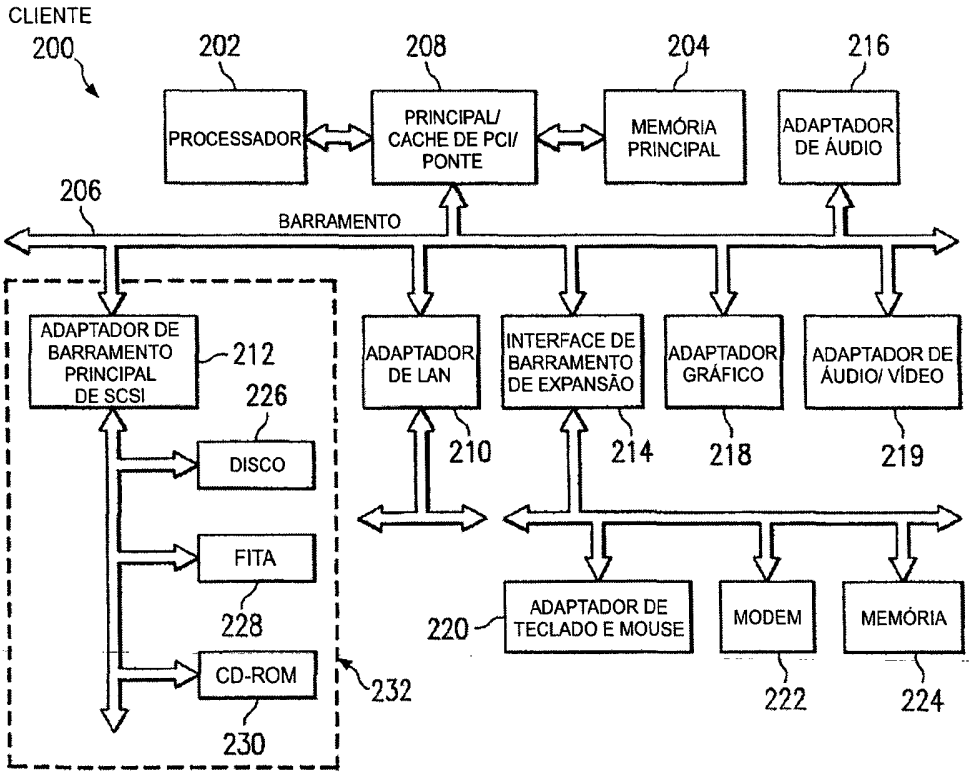


FIGURA 2

FIGURA 3

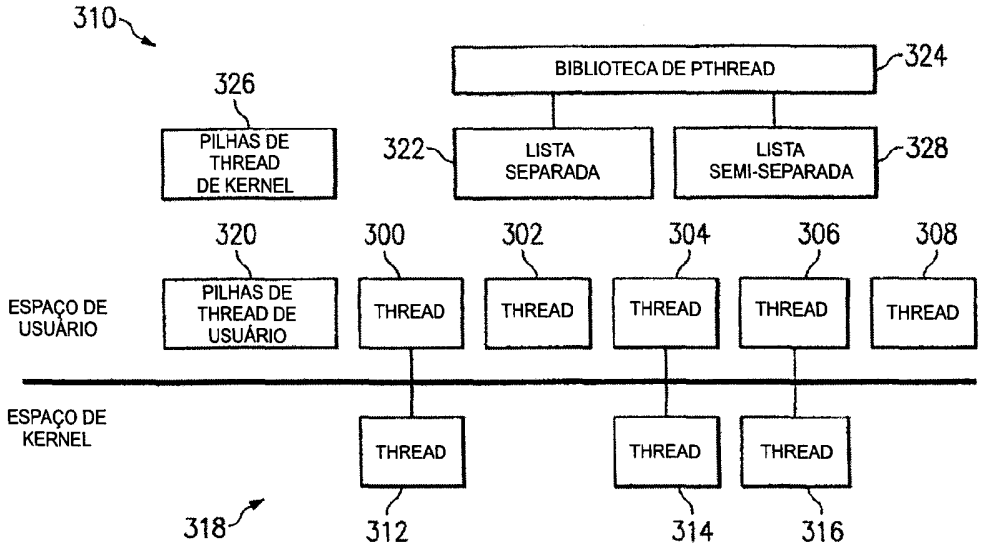
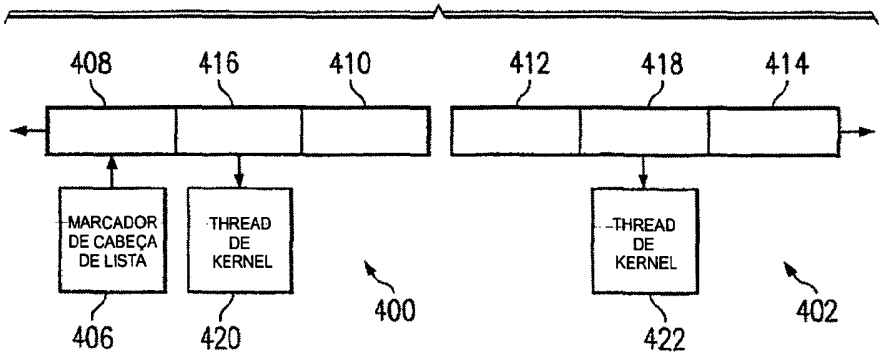


FIGURA 4A



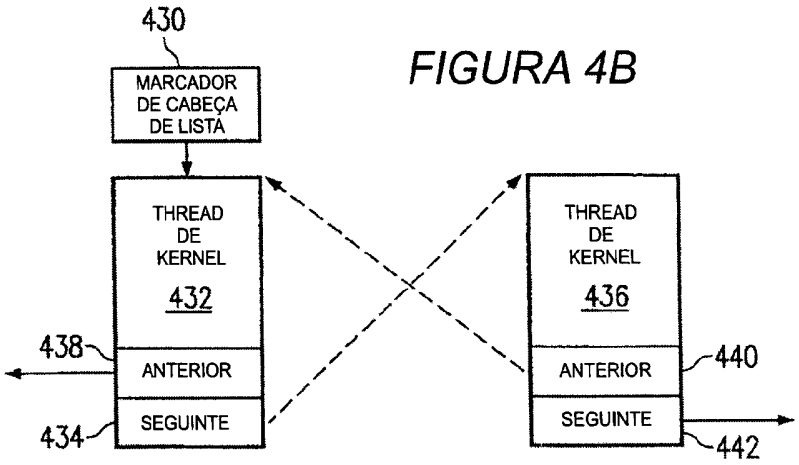


FIGURA 4B

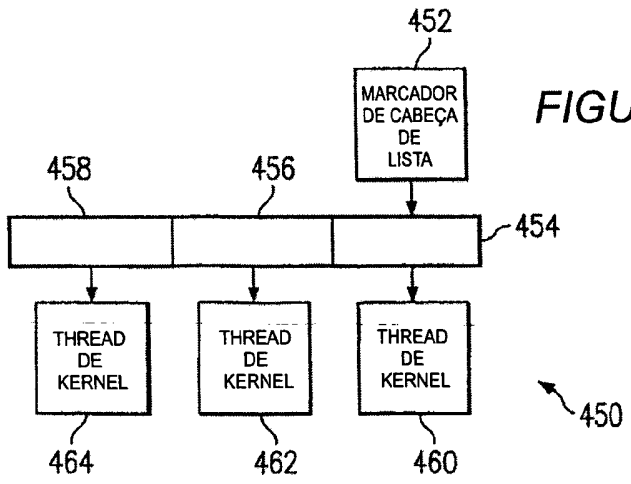


FIGURA 4C

FIGURA 5
(Técnica Anterior)

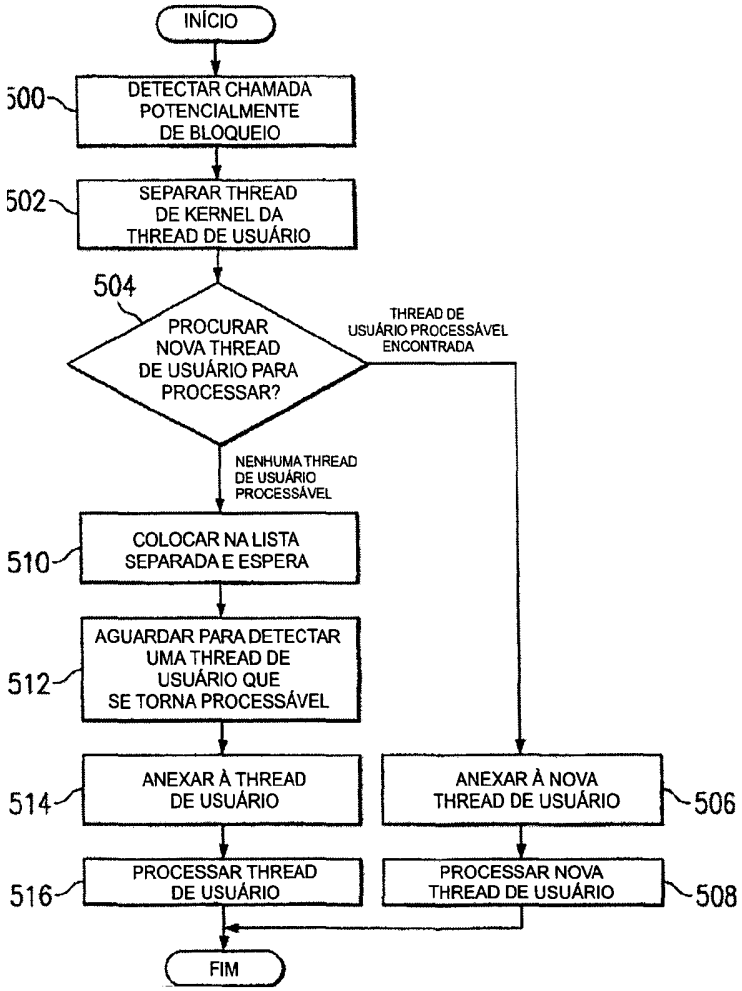


FIGURA 6

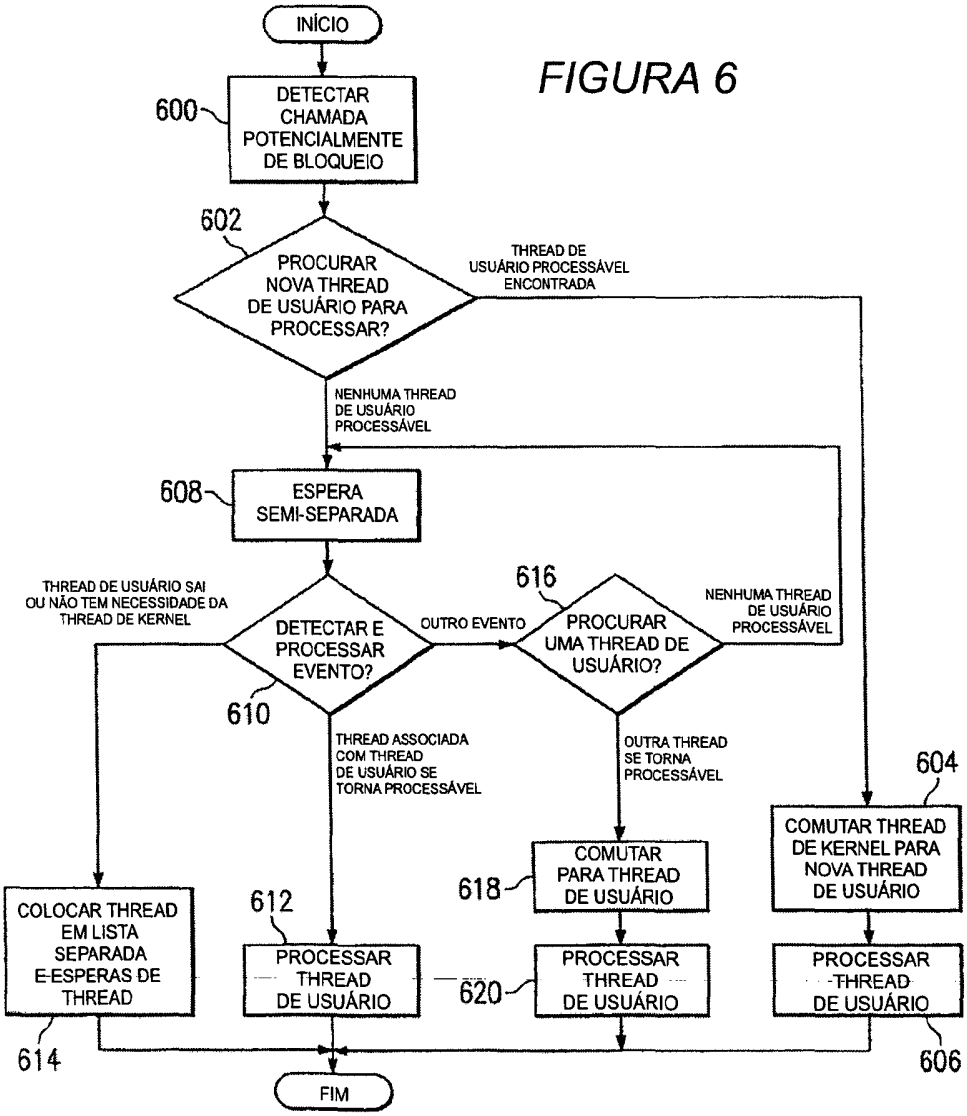

INÍCIO
600
DETECTAR CHAMADA POTENCIALMENTE DE BLOQUEIO
602
PROCURAR NOVA THREAD DE USUÁRIO PARA PROCESSAR?
THREAD DE USUÁRIO PROCESSÁVEL ENCONTRADA
NENHUMA THREAD DE USUÁRIO PROCESSÁVEL
608
ESPERA SEMI-SEPARADA
THREAD DE USUÁRIO SAI OU NÃO TEM NECESSIDADE DA THREAD DE KERNEL
DETECTAR E PROCESSAR EVENTO?
610
OUTRO EVENTO
616
PROCURAR UMA THREAD DE USUÁRIO?
NENHUMA THREAD DE USUÁRIO PROCESSÁVEL
THREAD ASSOCIADA COM THREAD DE USUÁRIO SE TORNA PROCESSÁVEL
OUTRA THREAD SE TORNA PROCESSÁVEL
604
COMUTAR THREAD DE KERNEL PARA NOVA THREAD DE USUÁRIO
618
COMUTAR PARA THREAD DE USUÁRIO
COLOCAR THREAD EM LISTA SEPARADA E ESPERAS DE THREAD
612
PROCESSAR THREAD DE USUÁRIO
620
PROCESSAR THREAD DE USUÁRIO
PROCESSAR THREAD DE USUÁRIO
614
606
FIM

FIGURA 7
(Técnica Anterior)

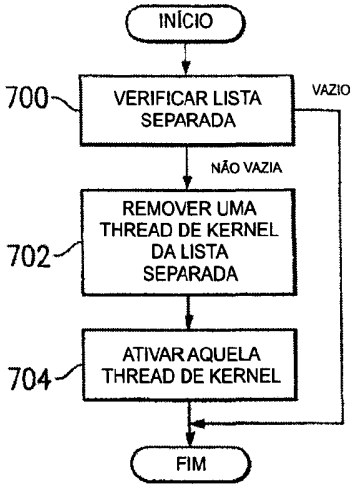
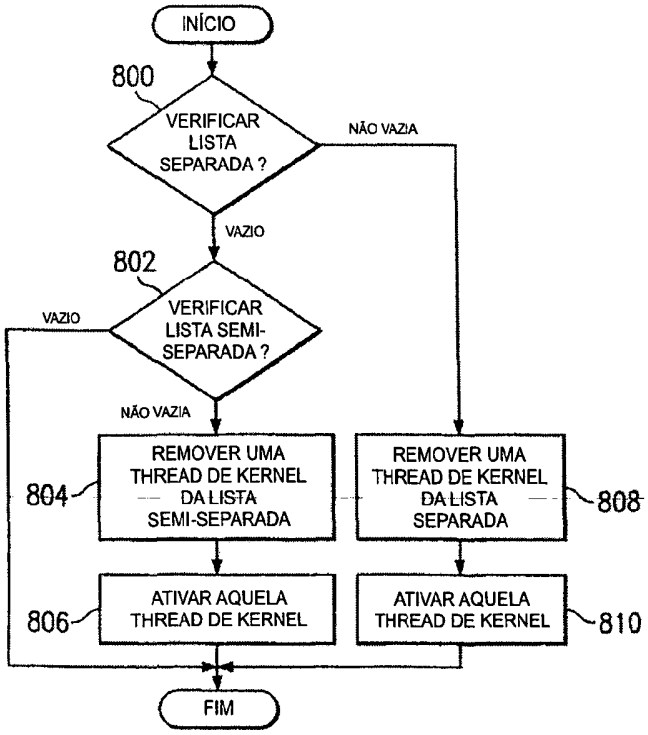


FIGURA 8



Resumo da Patente de Invenção para: **"MÉTODO PARA MANIPULAÇÃO DE THREADS EM UM SISTEMA DE PROCESSAMENTO DE DADOS E SISTEMA DE PROCESSAMENTO DE DADOS"**.

Método, aparelho e instruções para computador para gerenciamento de threads. Uma thread de kernel (312, 420, 432) associada com uma thread de usuário (300) é detectada como sendo desnecessária pela thread de usuário. A thread de kernel é parcialmente separada (328) em que dados para a thread não mudam pilhas em resposta à thread de kernel (312, 420, 432) ser desnecessária.