UNIDADE 2 – PROCESSOS E MEMÓRIA
MÓDULO 1 – PROCESSOS E TAREFAS

01

1 - GERENCIAMENTO DE PROCESSOS

Como comentado na unidade anterior, um processo pode ser pragmaticamente definido como a instância de um programa em execução. Um processo é, ainda, a unidade básica de execução em um Sistema Operacional, sendo que um único programa pode desencadear a execução de múltiplos processos.

Na prática, todas as tarefas executadas pelo processador, sob gerência do sistema operacional, são realizadas através de processos. Desta forma, como não é incomum ao usuário navegar por diversos aplicativos simultaneamente, o sistema operacional tem que lidar com o controle da execução de diferentes processos ao mesmo tempo.

Nos sistemas com múltiplos processadores, há a possibilidade de que cada processador se encarregue de um processo, o que permite que diversos processos possam estar em execução em um mesmo momento. Entretanto, em um ambiente computacional tradicional é comum, mesmo em sistemas com múltiplos processadores ou núcleos, que um mesmo processador se encarregue da execução de múltiplos processos simultaneamente.

Esta atividade, de gerenciar o uso do processador entre os diferentes processos garantindo a execução paralela de tarefas e o uso eficiente e eficaz dos recursos computacionais, é uma das principais atribuições dos sistemas operacionais modernos.

02

O Sistema operacional possui múltiplas formas para criação de novos processos. Tem-se que um processo está criado no SO quando a sua entrada é registrada no bloco de controle do processo (PCB) do Sistema Operacional e lhe é alocado espaço exclusivo para armazenamento em memória. Os principais eventos que ocasionam a criação de novos processos pelo sistema operacional são:

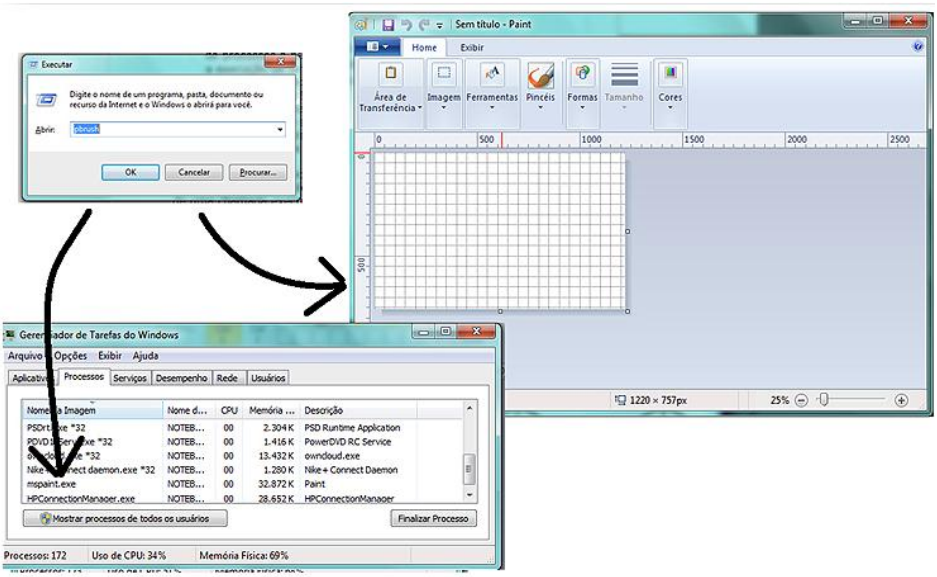
© 2015 - AIEC - Associação Internacional de Educação Continuada

Inicialização do Sistema Operacional →	Toda vez que o sistema operacional é iniciado múltiplos processos são automaticamente criados como, por exemplo, o processo responsável pela realização da autenticação do usuário ou o que gerencia os eventos do sistema.
Interação do usuário →	O usuário pode motivar a criação de uma série de processos a partir simplesmente da sua interação com o sistema. Quando inicia a execução de um software, por exemplo, o usuário está solicitando ao sistema operacional a criação de um processo que está relacionado ao funcionamento daquele aplicativo.
Rotina do Sistema Operacional →	O próprio sistema operacional é um agente que pode solicitar a criação de novos processos a partir de um processo já existente, de modo a tornar mais eficiente ou eficaz a execução de uma determinada tarefa.

Após a etapa de inicialização do sistema, independente do evento responsável pela criação, tem-se que um novo processo é gerado a partir de uma chamada executada por um outro já existente, seja ele de usuário ou de sistema. Os novos processos criados podem atuar de forma independente, ou seja, sem vinculação com qualquer outro processo, ou como um subprocesso inserido dentro de uma estrutura hierárquica, estando vinculado ao processo pai dentro de uma árvore de processos.

03

Tomando o Sistema Operacional Microsoft Windows como exemplo, um usuário poderia iniciar um novo processo clicando sobre o ícone executável do aplicativo na área de trabalho ou a partir da inserção do endereço do aplicativo no menu “executar” do sistema operacional. Na figura abaixo é possível observar a exibição do aplicativo “Paint” e a alocação do processo “mspaint.exe” como resultado da ação realizada pelo usuário ao executar o comando “pbrush” na caixa de execução do MS Windows.



© 2015 - AIEC - Associação Internacional de Educação Continuada

04

Como já exposto, ao iniciar um novo processo o sistema operacional automaticamente cria uma entrada no bloco de controle de processos.

O PCB é uma estrutura de dados que descreve e representa o processo para o sistema operacional e é composto por um conjunto de atributos relacionados ao processo e que podem ser classificados em:

a. bloco de identificação do processo;
b. bloco de informações do controle do processo;
c. bloco de informações do estado do processador.

O bloco de informações de **identificação do processo** se refere aos dados de identificação do processo, do seu criador e do utilizador. A maioria dos sistemas operacionais implementa a identificação do processo como um número inteiro único, o PID, que é utilizado, dentre outras finalidades, para auxiliar na alocação dos recursos computacionais para o processo. Já a identificação do processo criador e a do utilizador só é implementada em sistemas operacionais multiusuários e multitarefas.

As informações sobre o estado e o escalonamento do processo estão agrupadas no bloco de informações **de controle do processo**. Dados como a prioridade de execução, o tempo de CPU, uso dos recursos, comunicação entre processos e o estado atual do processo estão agrupados neste elemento.

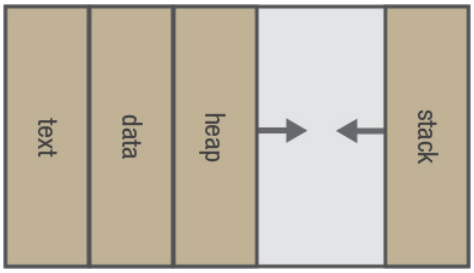
Já o bloco de informações do **estado do processador** agrupa os dados contidos nos registros do processador como, por exemplo, os de controle de estado.

05

2 - Modelo de Memória dos Processos

Ao gerar um novo processo, o Sistema Operacional, além de criar a entrada no PCB, automaticamente reserva o espaço de armazenamento na memória. Este espaço se torna, então, de uso exclusivo deste novo processo, evitando a concorrência pelo uso do recurso de memória compartilhado.

Quando alocado em memória, o processo é composto por uma série de elementos, cada um responsável pelo atendimento de uma necessidade específica. De forma simplificada, pode-se dividir o bloco de memória do processo em quatro diferentes seções: **text, data, heap** e **stack**.



© 2015 - AIEC - Associação Internacional de Educação Continuada

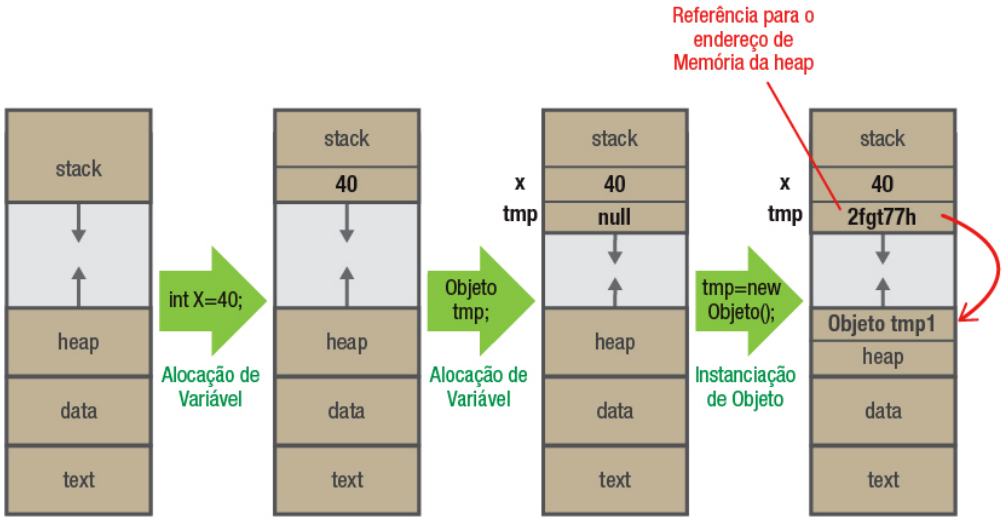
Neste modelo, o código a ser executado pelo processo é armazenado na seção **text** do bloco de memória. Normalmente o tamanho desta seção é fixo, tendo sido determinado durante a compilação do código fonte do programa. Já a seção **data** é o local onde estão armazenadas as informações relacionadas às variáveis globais e as variáveis locais estáticas, informações que, assim como o código a ser executado, podem ser definidas no momento da compilação do *software*, o que faz com que esta área também tenha tamanho fixo.

O tamanho do bloco de memória é justamente uma das principais características que diferenciam as seções heap e stack das demais, já que, diferente das seções text e data, o tamanho do bloco de memória alocado para estas áreas não é fixo, podendo variar de acordo com a execução do processo. Conforme pode ser observado no esquema de memória exibido na figura, todo o espaço livre do bloco alocado para o processo é utilizado de forma compartilhada pelas seções heap e stack.

06

A seção **stack**, ou pilha, é responsável por gerenciar todo o fluxo de execução do processo, empilhando e desempilhando chamadas de funções, parâmetros e variáveis locais que tem um tempo de vida fixo a depender do andamento da execução do programa. Já a seção **heap** é o local onde são armazenadas as variáveis dinâmicas como, por exemplo, a criação de uma instância de um objeto na linguagem de programação java ou a alocação de um bloco de memória através da utilização do comando *malloc* da linguagem de programação C.

O esquema de funcionamento das seções stack e heap pode ser visualizado na figura abaixo.



É relevante salientar que o conceito da divisão da memória em áreas é abstrato, já que não há a exigência que a área alocada para cada setor seja contígua, mas sim que sejam definidas áreas específicas para tratar de necessidades comuns, mesmo que fisicamente os blocos estejam fragmentados em memória. A forma como ocorre a alocação de memória irá depender do modelo de gerenciamento adotado pelo Sistema Operacional.

07

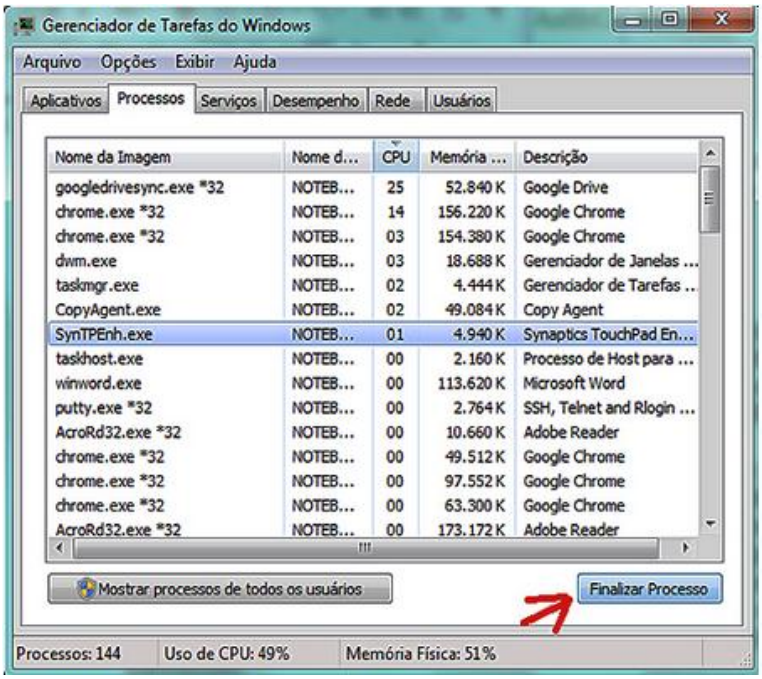
© 2015 - AIEC - Associação Internacional de Educação Continuada

3 - Finalização de Processos

Da mesma forma que ocorre com a criação, existem diversas maneiras de se finalizar um processo, algumas voluntárias e outras involuntárias. Adotando a lógica do caminho feliz, a maioria dos processos é finalizada voluntariamente quando termina com sucesso a execução da tarefa que lhe foi atribuída.

Outra saída voluntária é quando um erro conhecido é disparado pelo aplicativo. Por exemplo, imagine que, para que um determinado processo seja iniciado e executado corretamente, deve ser-lhe fornecido, como entrada, um arquivo em formato ".JPG". Mas, por um erro do usuário, um arquivo ".PDF" foi fornecido como entrada, o que ocasionou um erro na execução do aplicativo. Este é um exemplo de um término voluntário do processo a partir da ocorrência de um erro conhecido.

Já o término involuntário ocorre normalmente por uma falha na codificação do programa ou em razão da ação de um outro processo. A primeira causa está relacionada à execução de operações ou instruções ilegais pelo processo como, por exemplo, a realização de uma divisão por zero. Já a segunda é ocasionada por uma chamada de sistema que força a finalização forçada do processo.
No Microsoft Windows o próprio usuário pode solicitar a finalização forçada de um determinado processo a partir do gerenciador de tarefas do sistema operacional, conforme pode ser visualizado na figura.



Caminho feliz
Fluxo principal de execução de um processo, onde é esperado que o processamento seja realizado com sucesso sem a ocorrência de exceções ou erros.

08

4 - Estados de um Processo

A estrutura dos sistemas operacionais modernos trabalha com a existência de três diferentes estados de processo e quatro tipos de transições entre processos. Isto se faz necessário em ambientes multiprogramáveis onde o SO tem que gerenciar a execução simultânea de mais de um processo. A figura abaixo apresenta os três estados: **pronto, em execução,** e **em espera**, bem como as transações possíveis entre cada um deles.

© 2015 - AIEC - Associação Internacional de Educação Continuada

- Quando um determinado processo está no estado **“pronto”**, ou *ready*, significa que o mesmo está em condições de ser posto em execução, ou seja, está apenas aguardando ser alocado no processador para realizar a sua tarefa.
- Já o estado **“em execução”**, ou *running*, é alcançado quando o processo está efetivamente alocado na CPU.
- Por sua vez, o estado **“em espera”**, ou *wait*, é alocado a um processo que está a espera de uma resposta externa ou aguardando a liberação de recursos computacionais compartilhados para que possa prosseguir com o processamento.

09

Como já comentado, quatro transições são permitidas entre os três estados existentes. A transição entre os estados de pronto e o estado em execução, independentemente da direção da operação, é normalmente ocasionada pelo agendador de processos do sistema operacional e depende de critérios e algoritmos definidos especificamente para cada sistema operacional. A transição de um processo que está em execução para o estado de pronto pode ocorrer, por exemplo, porque este já esgotou o tempo alocado para execução no processador e o Sistema Operacional precisa liberar o recurso para que outro processo possa ser alocado. Já a transição inversa normalmente ocorre quando o espaço no processador foi liberado e, segundo as regras de agendamento do sistema, este processo tem a prioridade na execução.

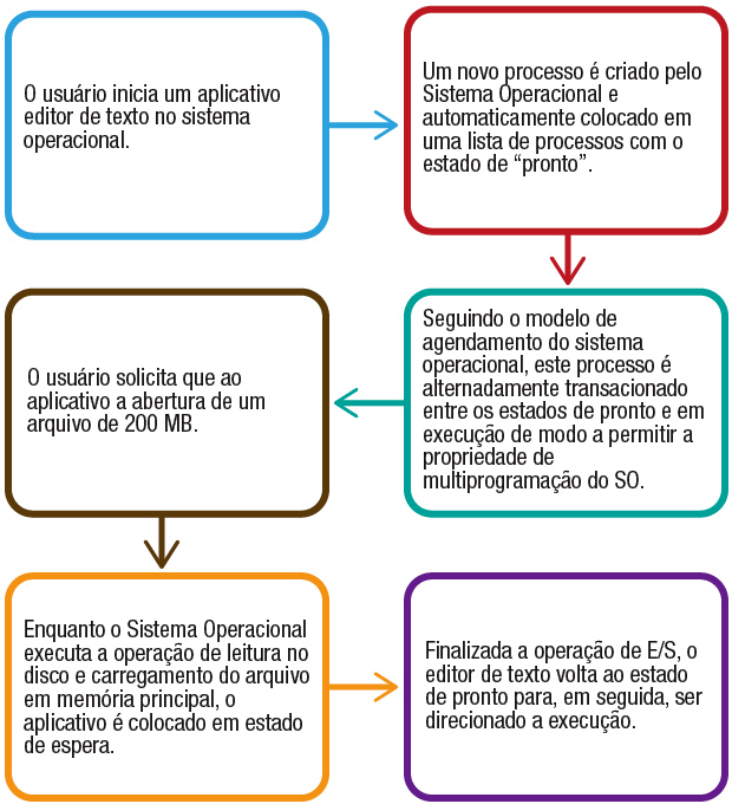
Já a transição do estado de execução para o de espera é ocasionada quando um processo, mesmo ainda tendo tempo disponível em CPU, não pode prosseguir por estar aguardando alguma resposta ou ação externa como, por exemplo, o resultado de uma operação de entrada e saída.

É importante ressaltar que a operação inversa não é permitida, ou seja, **a única transição permitida a partir do estado de espera é a movimentação para o estado de pronto**, que ocorre quando não existe mais a pendência para a execução de um determinado processo e o mesmo pode voltar para a fila de agendamento para execução.

10

Para esclarecer o fluxo entre os estados de um processo, imagine a seguinte sequência hipotética de ações:

© 2015 - AIEC - Associação Internacional de Educação Continuada



É interessante ressaltar que as trocas de contexto muitas das vezes ocorrem em décimos ou milésimos de segundo, de modo que a alternância entre processos é transparente para o usuário final. Outro fato relevante é que boa parte dos atuais microcomputadores possuem processadores com múltiplos núcleos, o que permite que mais de um processo seja executado ao mesmo tempo sem que exista a necessidade de mudança de contexto.

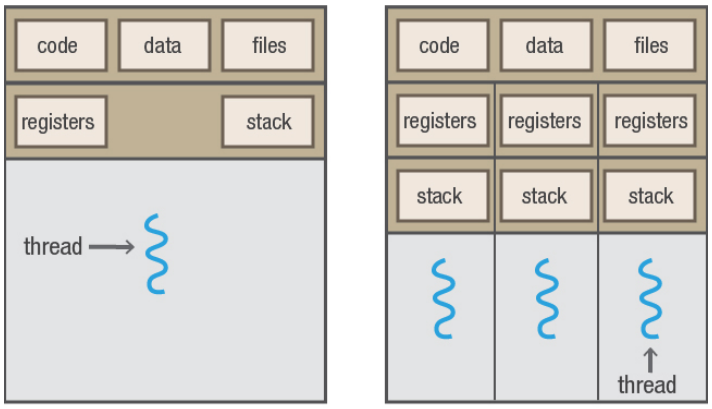
11

5 - Modos de execução de Threads

Já vimos de forma sucinta, na unidade anterior, o conceito de **thread, elemento utilizado como forma de reduzir o custo computacional decorrente das constantes trocas de contexto entre os processos**. Além de evitar as trocas, uma outra necessidade que motivou a utilização deste instrumento é o fato de que, em determinadas situações, é desejável que tarefas executadas em paralelo tenham acesso a uma mesma área de memória como forma de tornar mais ágil a realização do trabalho necessário.

Em seu conceito tradicional, as threads são executadas dentro do contexto de um processo, que pode ser single-threaded ou multithreaded, ou seja, permitir a execução de apenas uma thread ou de múltiplas threads simultaneamente por processo.

© 2015 - AIEC - Associação Internacional de Educação Continuada



Em ambientes **multithreaded** cada uma das threads compartilha o processador, da mesma forma que fazem os processos. Além disso, apesar de compartilharem a mesma área de memória, cada thread tem um contexto próprio de hardware. Em modelos **single-threaded** não há essa necessidade, já que não existe concorrência entre threads.

Outra similaridade do ambiente multithreaded com o esquema adotado nos processos é a existência do Bloco de controle da Thread (TCB), responsável por armazenar informações importantes em relação a thread, como o estado do processador, e em relação ao processo, como o espaço de endereçamento e os recursos disponíveis.

12

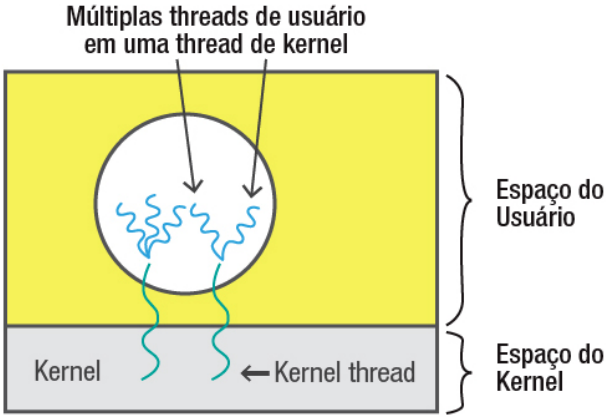
No contexto dos sistemas operacionais, as threads podem ser executadas tanto em **modo usuário** quanto em **modo kernel**, sendo possível, ainda, a **abordagem híbrida** e a utilização do modelo "Scheduler Activations".

MODO USUÁRIO	MODO KERNEL
Quando múltiplas threads são executadas no contexto de um único processo e **em modo usuário**, para o sistema operacional é como se apenas uma tarefa estivesse sendo executada. Desta forma, cabe à aplicação a implementação de toda a gerência e controle da concorrência das tarefas que estão sendo executadas em paralelo. Se por um lado esta abordagem se mostra desfavorável, por aumentar a complexidade no desenvolvimento das aplicações, por outro ela permite a execução de um software multithreaded mesmo que o sistema operacional não suporte esta característica. Outras vantagens deste modelo é o seu melhor desempenho e maior flexibilidade.	Já quando as theads são executadas **em modo kernel**, toda a complexidade de gerenciamento da concorrência é deixada sob responsabilidade do Sistema Operacional. Diferente do modelo executado em modo usuário, o SO sabe da existência de cada uma das threads e atua diretamente no agendamento da execução destas junto ao processador. A principal desvantagem desta abordagem é o baixo desempenho se comparado com o modelo executado em modo usuário, a principal vantagem, como já comentado, é a fácil integração com os serviços do sistema.

© 2015 - AIEC - Associação Internacional de Educação Continuada

13

A abordagem **híbrida** surgiu como uma forma de unir as vantagens da execução das threads em modo usuário e kernel. A implementação deste método permite que cada thread em modo kernel tenha múltiplas threads em modo usuário sendo executadas sobre a sua estrutura. O principal problema desta arquitetura é que não há uma comunicação entre as threads que estão em modo usuário e as que estão em modo kernel, desta forma, quando uma thread em modo kernel entra em estado de espera todas as threads vinculadas a esta automaticamente assumem o mesmo status.



Dos modelos existentes, o ***Scheduler Activations*** é o que melhor consegue agregar os benefícios de flexibilidade e desempenho encontrados na arquitetura de modo usuário com a facilidade de implementação inerente ao modelo de execução em modo kernel. Diferentemente do modelo híbrido convencional, o Scheduler Activations tem a capacidade de evitar transições desnecessárias entre as threads que estão sendo executadas em modo usuário e kernel, o que aumenta a performance do sistema. Isto se dá principalmente porque há uma comunicação efetiva entre a estrutura existente no modo usuário e o modo kernel do sistema operacional.

14

Resumo

Um processo é um programa em execução, sendo o seu modelo de gestão uma das principais tarefas dos sistemas operacionais modernos. Existem múltiplas maneiras utilizadas pelo Sistema Operacional na criação de novos processos, sendo a inicialização do SO, as interações do usuário e as rotinas do sistema operacional os principais eventos geradores.

Ao iniciar um novo processo, o sistema operacional automaticamente executa duas operações, cria uma entrada no bloco de controle de processos (PCB) e reserva o espaço de armazenamento deste processo na memória. O PCB é uma estrutura de dados que descreve e representa o processo para o sistema operacional e é composto por uma série de atributos relacionados ao processo, como a identificação do próprio processo e do seu criador.

© 2015 - AIEC - Associação Internacional de Educação Continuada

A finalização de um processo, assim como na criação, poder ser realizada de diferentes maneiras. Voluntariamente, um processo é finalizado quando termina com sucesso a execução da tarefa que lhe foi atribuída ou quando um erro conhecido é disparado pelo aplicativo. Já o término involuntário ocorre normalmente por um erro na codificação do programa ou por uma ação de um outro processo.

Durante o seu ciclo de vida, os processos podem assumir três diferentes estados: “pronto” – quando está em condições de ser posto em execução; “em execução” - quando está efetivamente alocado na CPU; e “em espera” – no momento que está aguardando uma resposta externa ou a liberação de recursos computacionais compartilhados para que possa prosseguir com o processamento. São permitidas três transições entre estes estados, entre os estados de pronto e em execução, independentemente do sentido, do estado de execução para o estado de espera e deste para o estado de pronto.

De forma a reduzir a quantidade de mudanças de estado entre os diferentes processos, operação que ocasiona perda de capacidade de processamento, foi introduzido no âmbito dos Sistemas Operacionais o conceito de threads. Em seu conceito tradicional, as threads são executadas dentro do contexto de um processo, que pode ser single-threaded ou multithreaded, ou seja, permitir a execução de apenas uma thread ou de múltiplas threads simultaneamente. No contexto dos sistemas operacionais, as threads podem ser executadas tanto em modo usuário quanto em modo kernel, sendo possível, ainda, a abordagem híbrida e a utilização do modelo “Scheduler Activations”.

UNIDADE 2 – PROCESSOS E MEMÓRIA

MÓDULO 2 – MULTIPROGRAMAÇÃO E COMUNICAÇÃO ENTRE PROCESSOS

01

1 – MULTIPROGRAMAÇÃO

Os computadores modernos têm uma característica em comum – permitir a execução de múltiplas tarefas ao mesmo tempo. Neste sentido, os diversos aplicativos que são executados para promover a realização destas tarefas são organizados em um número determinado de processos sequenciais. No contexto dos sistemas operacionais, este conceito é definido como multiprogramação.

Quando um novo aplicativo é codificado toda a complexidade de gestão da execução dos processos é abstraída, já que cabe ao sistema operacional a responsabilidade pela gerência do conjunto de processos através do controle do uso compartilhado e concorrente de recursos, como processador e memória.

Na prática, o conceito de multiprogramação pode ser observado pelo usuário quando este alterna entre programas que estão sendo executados através das janelas do Microsoft Windows. É comum usuários manterem abertos diversos programas simultaneamente, navegando na internet ao mesmo tempo em que mantém o leitor de e-mails ou o editor de texto em execução, por exemplo.

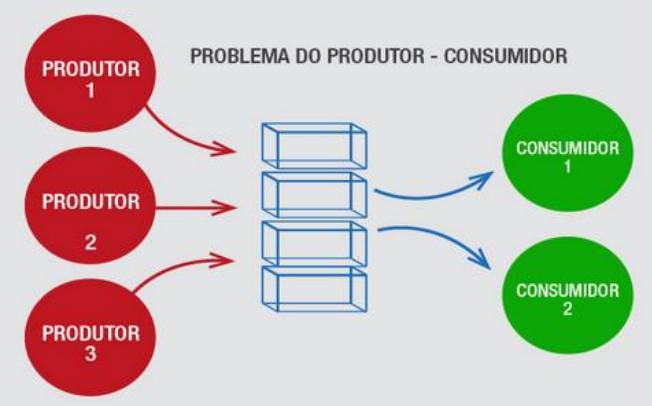
02

© 2015 - AIEC - Associação Internacional de Educação Continuada

2 - Compartilhamento de Recursos

No contexto da utilização de recursos compartilhados entre aplicações diferentes, é fundamental que exista alguma forma de comunicação que evite a disputa pelo uso dos recursos em processos que são executados de forma concorrente. Problemas no acesso a recursos compartilhados podem causar uma série de problemas, como a finalização involuntária dos aplicativos em execução ou até mesmo a geração de resultados incorretos após o processamento.

Uma parábola muito utilizada para exemplificar o problema no compartilhamento de recursos é a do produtor-consumidor. Imagine que dois processos compartilham um mesmo local para troca de informações, sendo que um processo, denominado produtor, gera os dados e grava na memória e o outro, chamado de consumidor, acessa o mesmo espaço de armazenamento para consumir os dados que foram gravados.



O que aconteceria se o produtor tentasse gravar uma nova informação num espaço de memória ainda ocupado? E qual seria o resultado se o consumidor tentasse ler o espaço compartilhado antes da gravação das informações?

Situações como esta, onde o resultado do processamento depende da sequência em que um determinado recurso compartilhado é acessado pelos processos, são denominadas **condições de corrida**.

03

Além dos problemas relacionados a condições de corrida, a competição por recursos pode, ainda, provocar outros diferentes problemas para o Sistema Operacional, como a inanição (*starvation*) e o *deadlock*.

A **inanição** ocorre quando o acesso a um determinado recurso é sempre negado a um dado processo que, por este motivo, nunca consegue finalizar a sua execução.

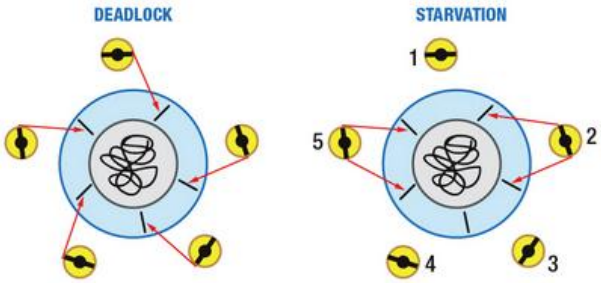
Já o **deadlock** é a situação onde dois ou mais processos estão esperando uns pelos outros para terminar a tarefa, de modo que nenhum deles consegue efetivamente completar o processamento.

Assim como na condição de corrida, um problema hipotético foi construído para exemplificar os conceitos de *deadlock* e *starvation*. Imagine agora que, em um determinado momento, tem-se "N" filósofos sentados ao redor de uma mesa circular comendo spaghetti e discutindo filosofia. Cada filósofo tem um prato de spaghetti e entre cada prato existe um garfo. Os filósofos comem e pensam alternadamente, mas, para comer, precisam utilizar dois garfos.

A grande questão a ser resolvida é permitir que todos os filósofos consigam comer, mesmo que não simultaneamente. Para isso, o algoritmo tem que evitar que os filósofos acabem pegando um único

© 2015 - AIEC - Associação Internacional de Educação Continuada

garfo ao mesmo tempo – *deadlock*, ou que apenas dois dos filósofos monopolizem quatro dos garfos e faça com que os outros três nunca tenham acesso a comida – *starvation*.



Observe na figura que na situação de *deadlock* nenhum dos filósofos consegue executar a tarefa por estarem à espera da liberação de recursos uns dos outros.

Já na situação de *starvation*, os filósofos de número 2 e 5 monopolizam o recurso impedindo o acesso dos filósofos 1, 3 e 4 que, desta forma, abaram por morrer de inanição.

04

3 - Exclusão Mútua

Uma das formas mais triviais de se solucionar as condições de corrida, deadlocks e starvation é a utilização da exclusão mútua. Este algoritmo evita o uso simultâneo dos recursos comuns através da utilização do conceito de região crítica.

Região crítica é a parte do código da aplicação responsável por realizar o acesso ao recurso compartilhado. A premissa adotada é a de que, independentemente da situação de momento do recurso compartilhado, se dois processos não acessarem simultaneamente as suas regiões críticas, a condição de corrida seria evitada.

As condições lógicas para a correta implementação de regiões críticas são:

- Dois processos não podem acessar simultaneamente a sua região crítica.
- Os processos não podem esperar para sempre para adentrar a sua região crítica
- Processos que estão executando fora de suas regiões críticas não podem bloquear o acesso de outro processo a sua região crítica
- Nenhuma suposição pode ser realizada a respeito da velocidade ou do número de CPUs.

05

Existem duas diferentes formas de implementação da lógica de exclusão mútua:

- os mecanismos com espera ociosa (*busy waiting*),
- os mecanismos com suspensão do processo de espera.

Os **processos com espera ociosa** se caracterizam na prática por, antes de entrar em sua região crítica, checar se a operação é permitida e, caso o acesso seja negado, iniciar um loop de verificações até que o acesso seja concedido.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Um dos aspectos negativos desta abordagem é o desperdício de tempo de processamento decorrente do processo cíclico de verificação. Como exemplos de soluções *busy waiting* pode-se citar a desabilitação de interrupções, a solução de Peterson, o uso de variáveis do tipo trava (*lock*), o chaveamento obrigatório (*strict alternation*) e a instrução TSL.

Além das soluções de espera ociosa, foram desenvolvidas uma série de outras alternativas para controle da concorrência sem que, no entanto, fosse necessário iniciar um processo de espera ativa, os chamados modelos com **suspensão do processo de espera**. A utilização de semáforos, monitores e troca de mensagens (*Message Passing*) são exemplos destas implementações.

06

4 - Técnicas de Espera ociosa

Existem diferentes formas de implementação da exclusão mútua através da utilização de métodos de espera ociosa, neste módulo serão analisadas:

- técnicas de desabilitação de interrupções,
- a solução de Peterson,
- o uso de variáveis do tipo trava,
- o chaveamento obrigatório e
- a instrução TSL.

A **desabilitação de interrupções** é uma alternativa de controle baseada em *hardware* onde cada processo que entra em sua região crítica desabilita automaticamente as interrupções do sistema, impedindo a troca de contexto entre os processos e, consequentemente, o acesso concorrente ao recurso compartilhado.

A grande desvantagem desta abordagem é o fato de que se um determinado processo, ao sair da região crítica, por algum motivo não habilitar as interrupções, todo o sistema sofrerá uma queda.

Além disso, em sistemas com mais de uma CPU, a desabilitação das interrupções atinge apenas uma das Unidades de Processamento, de modo que processos que estão sendo executados nas outras CPUs continuariam com acesso as suas regiões críticas e, por conseguinte, poderia vir a ocorrer um acesso simultâneo, ocasionando uma condição de corrida.

07

Diferentemente da desabilitação de interrupções, a solução de **variáveis do tipo trava** é baseada em *software* e não em *hardware*.

A ideia se fundamenta na criação de uma variável que é iniciada com o valor 0. Sempre que um processo tem a necessidade de adentrar em sua região crítica, ele verifica o estado atual da variável e:

- Se o valor é 0; o processo altera o valor para 1 e entra em sua região crítica.

© 2015 - AIEC - Associação Internacional de Educação Continuada

- Se o valor é 1; o processo aguarda até que se torne 0.
- Ao sair da região crítica, o processo deve retornar o valor da variável para 0.

A estratégia de bloqueio de variáveis tem uma grande falha. Imagine, por exemplo, que um determinado processo verificou o estado da variável e este estava definido como “0”. Só que no milésimo de segundo entre a verificação do valor da variável e a sua alteração para 1, um outro processo teve a necessidade de também adentrar em sua região crítica e também efetuou uma consulta a variável de controle, obtendo o valor 0.

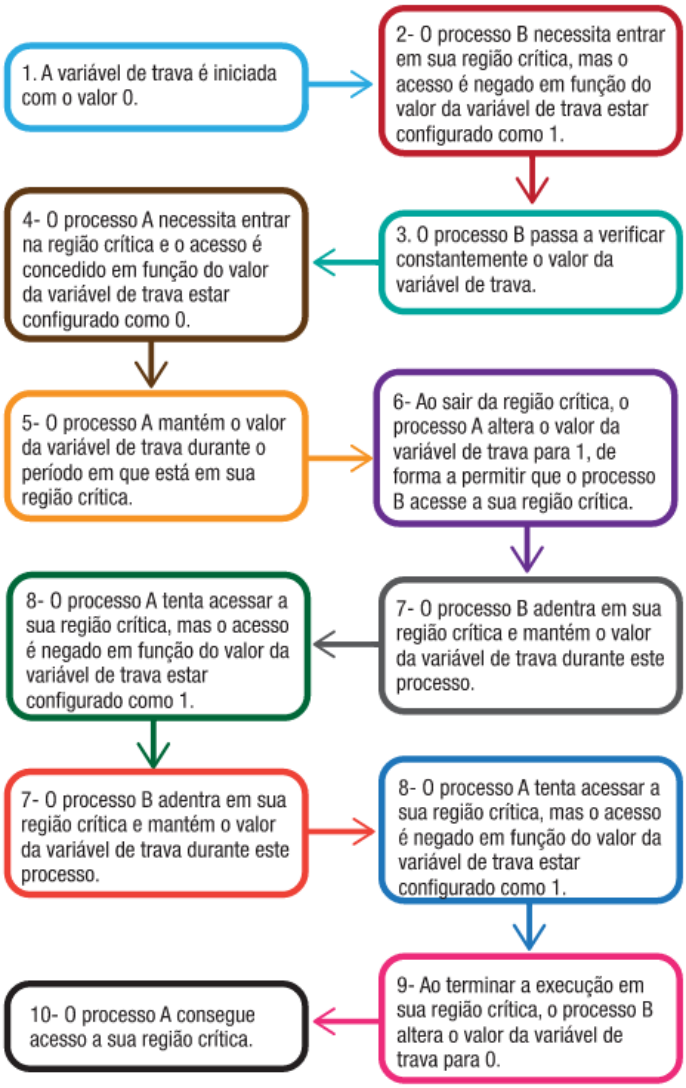
Nesta situação, os dois processos acessariam as suas regiões críticas ao mesmo tempo.

08

O **chaveamento obrigatório** é uma versão modificada do uso de variáveis do tipo trava, pela lógica deste algoritmo os diferentes processos esperariam diferentes condições da variável global, o que implicaria em uma constante alternância entre os processos e, por conseguinte, evitaria a condição de corrida.

Imagine uma situação onde para que o processo A possa entrar na sua região crítica é necessário que o valor da variável global seja igual a 0, ao tempo que para que o processo B possa entrar a variável deve estar com o valor 1. Com base neste contexto inicial, observe a seguinte sequência de ações:

© 2015 - AIEC - Associação Internacional de Educação Continuada



A grande desvantagem deste método, além do desperdício de processamento em função da espera ociosa, é que se espera que os diferentes processos tenham uma necessidade de acesso similar a suas áreas críticas. Em situações em que um determinado processo A necessita de muito mais acesso a sua região crítica do que o processo B, ele pode ser impedido de entrar em sua região crítica mesmo o outro processo não estando em sua região crítica em um dado momento.

09

Já o algoritmo de **Peterson** é uma simplificação do modelo de Dekker que, com base na junção dos métodos de variável de trava e chaveamento obrigatório, desenvolveu a primeira solução que resolve plenamente o problema da exclusão mútua.

O modelo utiliza duas variáveis, uma para registrar o interesse do processo em acessar a região crítica e outra para controlar a concorrência pelo uso do recurso. A lógica é que, ao decidir entrar na região

© 2015 - AIEC - Associação Internacional de Educação Continuada

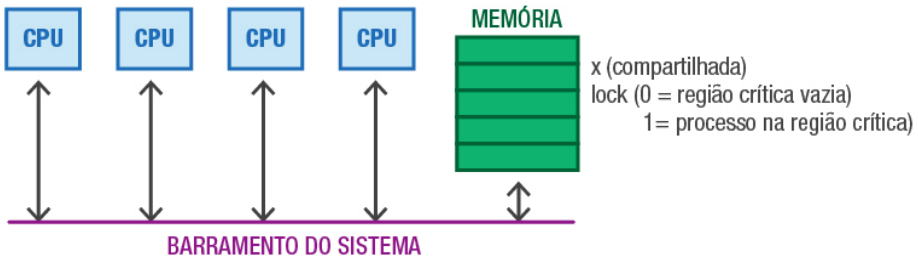
crítica, o processo manifesta o seu desejo, mas cede a vez para que o outro processo interessado tenha a prioridade no acesso.

Além de garantir a exclusão mútua, o método de Peterson impede que um dos processos fique aguardando para entrar em sua região crítica mesmo quando não há nenhum outro processo executando nesta região. O algoritmo de Peterson, em linguagem C, é exibido abaixo.

```
#define FALSE 0
    #define TRUE 1
    #define NPROCESSOS 2 /*Número de Processos*/

    int vez; /*Variável que controla de quem é a vez*/
    int interessado[NPROCESSOS]; /*controla o interesse dos processos em
entrar
    na região crítica, inicialmente nenhum processo tem interesse*/

    void entrar_regiao(int processo){
    int outroProcesso; /*Número do processo que está na vez*/
        outroProcesso = 1 - processo;
        interessado[processo] = TRUE; /*demonstra interesse em adentrar a
região crítica*/
        vez = processo;

    while (vez == processo && interessado[outroProcesso] == TRUE) /*Espera
ociosa*/
        região_critica(); /*acessa a região crítica*/
        }

    void sair_regiao(int processo){
        interessado[processo] == FALSE; /*Sinaliza que saiu da região
crítica*/
        }
    }
```

10

A última solução de espera ociosa é a **instrução TSL**, técnica baseada em *hardware* e que funciona apenas em computadores cujos processadores provêm a instrução “Test and Set Lock”. Esta função realiza duas atividades, primeiro lê o conteúdo da palavra de memória lock no registrador e, em seguida, armazena um valor diferente de zero no endereço de memória lock.



© 2015 - AIEC - Associação Internacional de Educação Continuada

A garantia de que apenas um processo fará esta operação por vez se dá em função da instrução ser indivisível, ou seja, a CPU que executou a chamada bloqueia o barramento de memória impedindo o acesso de outras CPUs a este espaço de memória até que a instrução esteja completada.

11

5 - Outras Técnicas para Exclusão Mútua

Das cinco soluções de espera ociosa apresentadas, apenas duas atendem completamente os requisitos necessários para prover de forma eficaz a exclusão múltipla – o **método de Peterson** e a **instrução TSL**. Apesar disto, estas não são soluções ótimas em função do processamento perdido por conta do custo da espera ociosa. Em função deste problema, grande parte dos sistemas operacionais tem o seu processo de controle baseado na utilização de semáforos, monitores e troca de mensagens.

A utilização de **semáforos** foi proposta por Dijkstra na década de 1960 como uma das soluções para o problema da exclusão mútua e da sincronização condicional dos processos.

Um semáforo é uma variável protegida que só pode ser acessada e alterada através de três operações:

- **inicialização**,
- **DOWN** (decrementa a variável) e
- **UP** (incrementa a variável).

As operações DOWN e UP são atômicas e indivisíveis e a variável do semáforo não aceita valores negativos. Desta forma, sempre que uma operação DOWN é realizada em um semáforo que tem valor 0 o processo é movido para o estado de espera. Saiba+

A grande vantagem do uso dos semáforos é que o processo não precisa ficar a todo o tempo verificando se o valor do semáforo foi alterado, já que este trabalho seria desempenhado pelo Sistema Operacional, sem gerar overhead. Assim que o processo em execução na região crítica informa ao SO que saiu desta região, através da chamada da função UP, o Sistema Operacional se encarrega de selecionar automaticamente um dos processos da fila de espera e de alterar o seu estado para “em execução”.

Saiba+
A versão simplificada dos semáforos, que trabalha com variáveis binárias ao invés de inteiras, é a variante normalmente utilizada para tratar do problema de exclusão múltipla. Nesta lógica, seria estabelecido um semáforo para cada recurso compartilhado, sendo que um processo chamaria a função DOWN quando tivesse a intenção de acessar a sua região crítica. Caso o valor armazenado no semáforo fosse igual a 1, este seria decrementado e o programa teria acesso à região crítica, se o valor fosse 0, o processo que fez a solicitação entraria em espera e o valor da variável de semáforo seria mantido.

Overhead

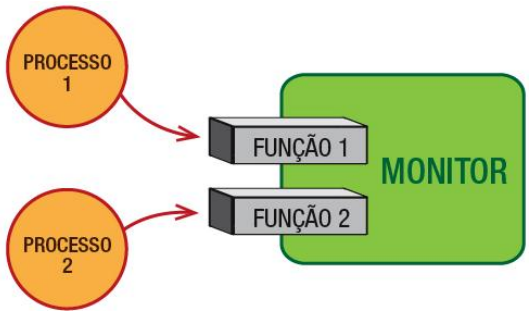
© 2015 - AIEC - Associação Internacional de Educação Continuada

O custo computacional relacionado à execução de uma determinada operação como, por exemplo, o gerenciamento de processos do sistema operacional, mas que não está diretamente relacionado a execução da tarefa solicitada pelo usuário e que produz o resultado final esperado.

12

Os **monitores** são uma construção das linguagens de programação que provêm função equivalente à fornecida pelos semáforos, só que de mais fácil controle. O conceito de monitor foi proposto por Hansen e Hoare na década de 1970 como uma forma de reduzir os erros de programadores na implementação de semáforos.

A solução para exclusão mútua deste método é baseada nas rotinas da própria estrutura do monitor, não sendo necessário esforço de codificação por parte do programador. Como apenas um processo pode executar um determinado procedimento do monitor em um dado momento, se algum outro processo efetuar a chamada, o próprio monitor já identifica e realiza o controle, ordenando os processos em uma lista de espera e mudando o seu estado paulatinamente à medida que o processo atualmente em execução termina a sua tarefa.



É importante ressaltar que um monitor pode possuir diversos procedimentos implementados, sendo que estes podem ser executados simultaneamente, desde que apenas um processo execute cada um dos procedimentos. Isso se dá porque cada procedimento está associado a uma diferente região crítica.

Monitor
Um monitor é uma estrutura de sincronização que requisita e libera a seção crítica associada a um recurso de forma transparente, sem que o programador tenha de se preocupar com isso. Esse conceito foi proposto em 1972 pelos cientistas Per Brinch Hansen e Charles Hoare.

13

As **trocas de mensagens** são um artifício provido pelo Sistema Operacional como forma de viabilizar a comunicação entre processos sem a necessidade do uso de variáveis compartilhadas.

Esta capacidade surge como um artifício capaz de sanar a dificuldade de se prover a exclusão mútua e o sincronismo em sistemas distribuídos, ou seja, computadores geograficamente espalhados e que, por conseguinte, não tem acesso a um espaço de memória compartilhado.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Este método provê ao menos duas operações, uma para enviar e outra para receber as mensagens. Além disso, se dois processos desejam manter uma comunicação através de troca de mensagens, uma ligação deve ser estabelecida, podendo a comunicação ser síncrona ou assíncrona, direta ou indireta.

14

RESUMO

Apesar dos computadores modernos permitirem a execução de múltiplas tarefas ao mesmo tempo, toda a complexidade da gestão destes processos é abstraída pelo Sistema Operacional, a quem cabe a responsabilidade pela gerência do conjunto de processos através do controle do uso compartilhado e concorrente de recursos como processador e memória. Neste contexto da multiprogramação, é fundamental que exista uma comunicação entre os processos de modo a evitar erros decorrentes da disputa pelo recurso compartilhado, como condições de corrida, starvation ou deadlock. Diversas são as técnicas utilizadas para contornar estes problemas, de sincronização de processos e compartilhamento de recursos, sendo uma das mais triviais a exclusão mútua, modelo que evita o uso simultâneo dos recursos comuns através da utilização do conceito de região crítica.

Os algoritmos de implementação da exclusão mútua se agrupam em dois blocos que se caracterizam por possuir ou não espera ociosa (*busy waiting*). Os processos com espera ociosa se caracterizam por verificarem ativamente e em loop se lhe é permitido o acesso a região crítica, operação que causa desperdício de tempo de processamento decorrente do processo cíclico de verificação. Já os modelos de suspensão do processo de espera são implementados sem a necessidade de gasto computacional com a verificação cíclica do acesso a região crítica.

Como exemplos de soluções *busy waiting* pode-se citar a desabilitação de interrupções, a solução de Peterson, o uso de variáveis do tipo trava (*lock*), o chaveamento obrigatório (*strict alternation*) e a instrução TSL. Já como exemplo das soluções que não efetuam a verificação em loop tem-se o uso dos semáforos, os monitores e a política de troca de mensagens.

Das cinco soluções de espera ociosa apresentadas, apenas duas atendem completamente os requisitos necessários para prover de forma eficiente e eficaz a exclusão múltipla – o método de Peterson e a instrução TSL. Apesar disto, estas não são soluções ótimas em função do processamento perdido decorrente do custo da espera ociosa. Todas as demais soluções alternativas apresentadas e que não implementam o loop de verificação de entrada ema região crítica atendem plenamente os requisitos de exclusão mútua.

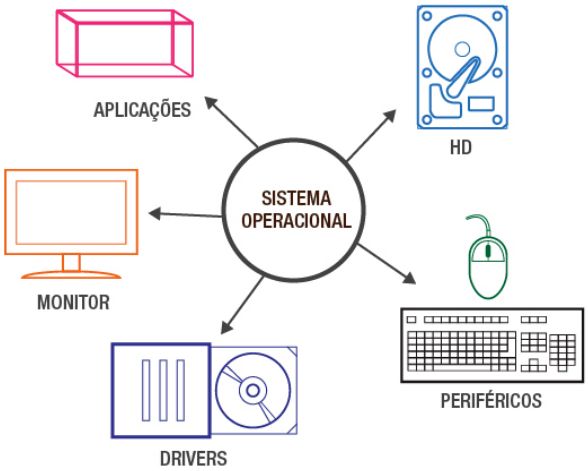
© 2015 - AIEC - Associação Internacional de Educação Continuada

01

1 - AGENDAMENTO DE PROCESSOS

Uma das principais características dos sistemas operacionais modernos é gerenciar os recursos computacionais e abstrair toda a complexidade envolvida nesta atividade, facilitando a tarefa do desenvolvimento de aplicativos.

Atualmente, o usuário tem a percepção de que é possível executar uma série de tarefas simultaneamente no seu desktop ou notebook, isto se dá porque é possível para o usuário operar o navegador de internet ao mesmo tempo em que escuta uma música no media player e concorrentemente ao fato de que o software antivírus efetua a varredura por arquivos infectados no computador.



Toda a gerência deste complexo ambiente multiprogramável, onde diversos processos são executados dentro de um mesmo intervalo temporal, é executada pelo Sistema Operacional do computador. Para realizar efetivamente esta atividade de gerenciamento de processos, o SO se baseia em uma série de algoritmos para promover o **agendamento** ou **escalonamento dos processos**.

02

Esta área de estudo é responsável por definir regras para determinar qual deve ser o processo posto em execução a partir de um grupo de processos prontos para serem executados. Para a execução desta tarefa, foram desenvolvidos diversos algoritmos baseados em diferentes paradigmas.

Os algoritmos de agendamento podem ser:

- preemptivos ou
- não preemptivos.

No caso dos modelos **não preemptivos**, quando um dado processo está em execução nenhum evento externo tem a capacidade de interromper o seu processamento e promover a alocação de um novo

© 2015 - AIEC - Associação Internacional de Educação Continuada

processo.

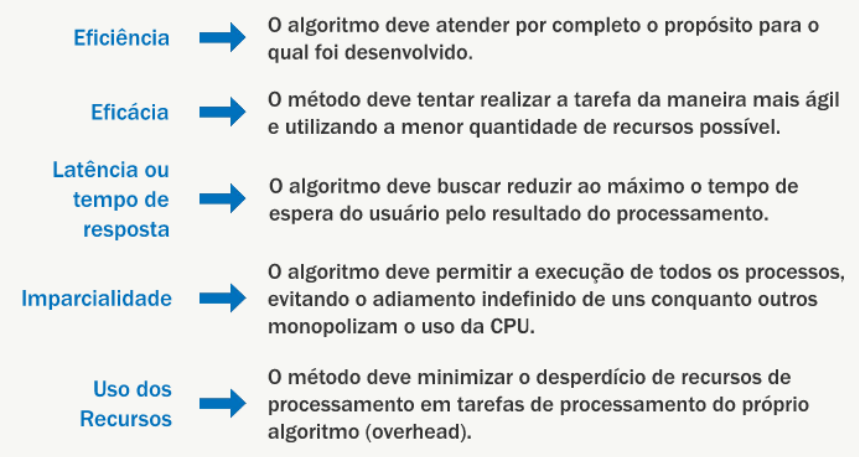
A única possibilidade de que o processo seja removido do procedimento de execução é uma alteração de estado promovida pelo próprio processo como, por exemplo, quando este necessita de algum elemento de entrada e modifica a sua situação para “em espera”.

Já os algoritmos **preemptivos** se caracterizam por estarem sujeitos a intervenções do Sistema Operacional no seu controle de execução dos processos. Desta forma, interrupções externas ocasionadas pelo SO tem o poder de alterar o estado do processo colocando-o, por exemplo, em espera ou em execução.

A maioria dos Sistemas Operacionais atuais utilizam métodos de escalonamento preemptivos.

03

Como já comentado, diversos são os modelos propostos para promover a implementação das políticas de agendamento do SO. De modo a avaliar o bom desempenho destes algoritmos, foram definidos alguns critérios, dentre estes, pode-se citar:



Nas próximas seções serão analisados oito dos principais algoritmos de escalonamento utilizados nos Sistemas Operacionais:

- First-In-First-Out;
- Shortest-Job-First;
- Round Robin;
- Por Prioridades;
- Múltiplas Filas;
- Fair-share;
- Loteria;
- Tempo Real.

04

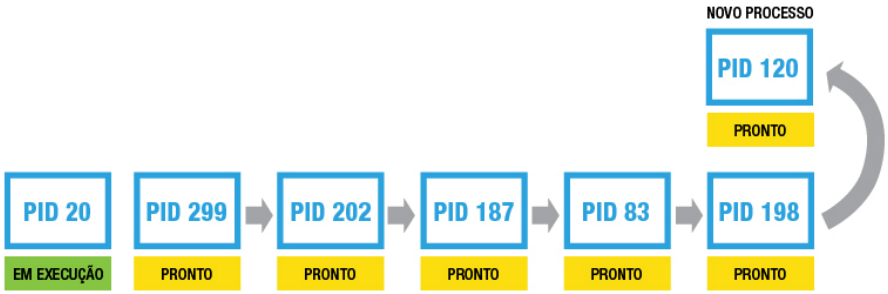
© 2015 - AIEC - Associação Internacional de Educação Continuada

2 - First-In-First-Out (FIFO)

Esta é a política de escalonamento que possui uma das regras mais simples dentre todas as existentes:

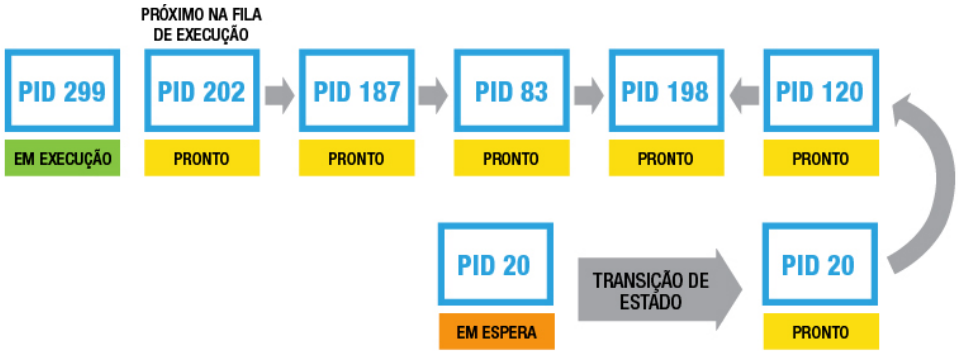
A lógica aplicada no algoritmo FIFO é a de que a ordem de execução dos processos obedecerá à ordem de chegada dos mesmos na estrutura de espera do processador, daí o nome First-In-First-Out ou *primeiro a entrar*, *primeiro a sair*.

Desta forma, o Sistema Operacional cria uma fila de processos em estado de "pronto" e, à medida que o processo que está em execução termina a sua tarefa, o próximo da fila automaticamente é alçado para o estado "em execução" e inicia o seu processamento.



05

Ainda segundo a lógica deste algoritmo, sempre que um processo for transacionado para o estado de "em espera" ele é removido da fila de execução e é retornado ao final da fila, assim que o seu estado retornar para "pronto".



O método de agendamento FIFO é não preemptivo, sendo comumente utilizado em sistemas que executa processamento em batch.

06

3 - Shortest-Job-First (SJF)

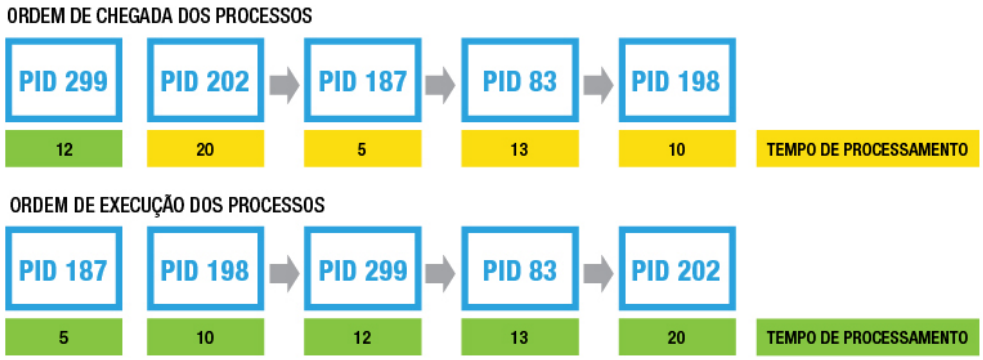
O método Shortest-Job-First ou *Processo mais curto primeiro* é um algoritmo de escalonamento que

© 2015 - AIEC - Associação Internacional de Educação Continuada

executa, dentre processos igualmente importantes, o mais curto primeiro.

Esse método parte do pressuposto que o tempo necessário para executar uma tarefa é conhecido antes do início do seu processamento.

Como na prática esta informação normalmente não é conhecida com antecedência, o sistema se baseia na estatística do histórico de execução de processos similares. Desta forma, é possível alocar primeiro os processos que levam menos tempo para execução, mesmo que não sejam os primeiros na fila de chegada.



07

A principal **vantagem** da utilização do método SJF é a redução do tempo médio e máximo de espera para execução dos processos. Como base de comparação, utilizando o algoritmo FIFO, onde os processos são executados a partir da ordem de chegada na fila de processamento, o tempo máximo de espera para o exemplo exposto na figura anterior seria de 50, e o tempo médio de 26,2, conforme pode ser observado na tabela de cálculo apresentada abaixo.

Processo	Tempo de Espera para execução
299	0
202	12
187	32
83	37
198	50
	Tempo de espera Médio: 26,2

Já com a utilização do método SJF, para os mesmos processos utilizados no cálculo anterior, o tempo máximo de espera é reduzido para 40 e o tempo médio cai parar 17,4, conforme exibido na tabela de cálculo abaixo.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Processo	Tempo de Espera para execução
187	0
198	5
299	15
83	27
202	40
	Tempo de espera Médio: 17,4

Assim como o FIFO, o algoritmo SJF é não preemptivo e comumente é utilizado como política de agendamento de sistemas em batch.

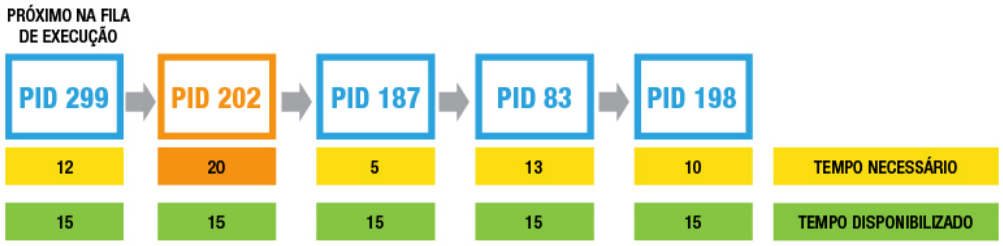
08

4 - Algoritmo Round Robin

O algoritmo de agendamento **Round Robin** é bastante similar ao FIFO, já que a fila de execução é ordenada de acordo com a posição de chegada dos processos.

A grande diferença se dá pelo fato de que, neste algoritmo, o processo nem sempre é executado por completo em seu primeiro ciclo de execução, já que, ao entrar na fila de execução, lhe é fornecido um tempo de uso da CPU que pode ser menor do que o tempo necessário para realização completa da tarefa.

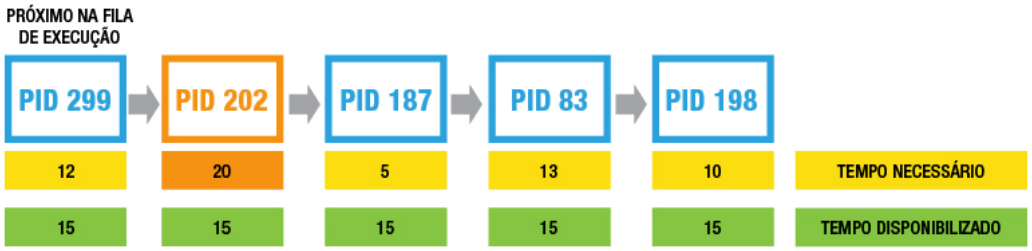
A figura abaixo representa uma fila de execução composta por cinco processos, sequenciados pela ordem de chegada. As caixas em verde representam o tempo disponibilizado pelo algoritmo para execução de cada um dos processos e as caixas em amarelo ou vermelho representam o tempo estimado que cada processo levaria para executar por completo a sua atividade.



09

Observando o exemplo, percebe-se que quatro dos cinco processos têm tempo de execução menor do que o tempo disponível e, por conseguinte, conseguiriam completar toda a tarefa em apenas um ciclo de processamento.

© 2015 - AIEC - Associação Internacional de Educação Continuada



Entretanto, o processo 202 possui um tempo de execução 5 ms maior do que o tempo disponibilizado para sua execução. Desta forma, ao esgotar o tempo fornecido, o Sistema Operacional interromperia a execução do processo, o colocando para o final da fila e liberando o próximo da fila para entrar em execução. Esta política realizada pelo Round Robin impede que um único processo monopolize o uso da CPU.

Como pôde ser observado, o método Round Robin permite que o Sistema Operacional interfira na política vigente de agendamento e execução dos processos através do uso de interrupções, o que caracteriza este algoritmo como preemptivo.

10

5 - Algoritmo por Prioridades e de Múltiplas Filas

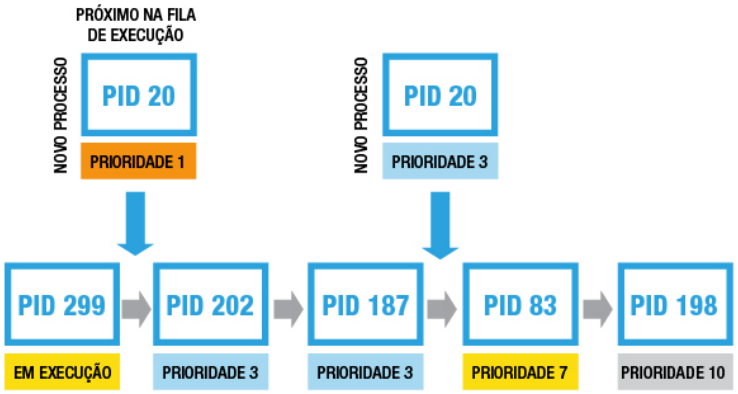
Os três algoritmos apresentados anteriormente têm uma característica em comum, tratam os processos como se todos tivessem a mesma prioridade, ou seja, não há qualquer privilégio para processos iniciados por um determinado usuário ou por um serviço de sistema.

Entretanto, existem situações que podem exigir **que determinados processos tenham prioridade sobre outros**. Tomando como estudo de caso uma empresa, seria possível, por exemplo, a definição de que processos demandados por usuários com perfil de diretor tivessem prioridade sobre processos iniciados por outros usuários da corporação.

Desta forma, utilizando o algoritmo por prioridades, no momento da criação de cada processo lhe seria atribuído um valor de prioridade, de modo que o algoritmo de agendamento pudesse identificar a posição onde o processo seria alocado na fila de execução.

No caso de processos com mesmo grau de prioridade, a regra de posicionamento para execução seria similar à estabelecida no método FIFO, ou seja, os processos seriam alocados após o último processo que tem o mesmo nível de prioridade.

© 2015 - AIEC - Associação Internacional de Educação Continuada



O agendamento por prioridades permite a adoção tanto de uma estratégia preemptiva quando não preemptiva. Na primeira opção, a chegada de um processo de maior prioridade causaria uma interrupção imediata da execução do processo de menor prioridade e a consequente mudança de contexto entre os processos. Na segunda alternativa, exibida na figura acima, o processo de maior prioridade aguardaria até o término da execução do processo atualmente alocado no processador para só então ser posto em execução.

11

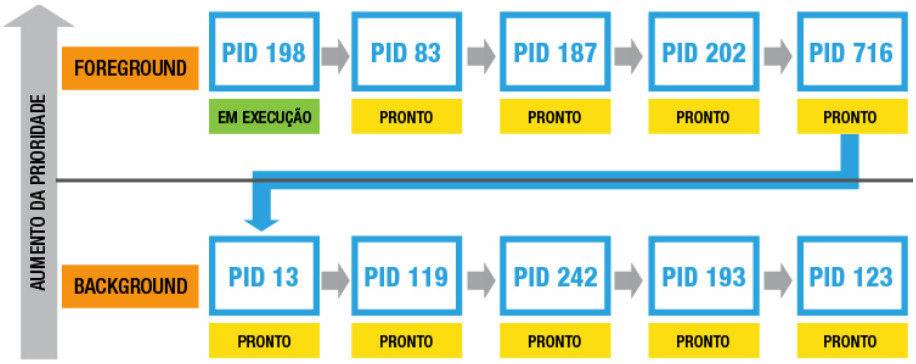
Múltiplas Filas

A implementação deste algoritmo preemptivo ocorre através da criação de múltiplas filas de modo a agrupar os processos por ordem de prioridade. Desta forma, as solicitações de menor prioridade só seriam executadas no caso de não existirem processos nas filas de maior prioridade.

O modelo de agendamento por múltiplas filas não deixa de ser um algoritmo baseado na priorização de processos, entretanto, diferentemente do algoritmo por prioridade, que associa um nível para cada processo, o de múltiplas filas **permite a criação de grupos de processos com características próprias** e possibilita a definição da prioridade pela sua importância para execução do aplicativo.

Como exemplo, um determinado SO poderia ter duas filas, uma apenas para o grupo de processos que requerem ações do usuário (*foreground*) e outra para os processos que são executados em plano de fundo pelo sistema operacional (*background*). Como as ações de usuário sempre demandam resposta rápida, a fila de processos de foreground teria prioridade sobre a de background.

© 2015 - AIEC - Associação Internacional de Educação Continuada



Com este modelo, mesmo que existisse uma infinidade de processos de sistema (*background*) na fila de execução, como a instalação de uma atualização ou até mesmo a realização de uma varredura por vírus, os processos iterativos de usuário (*foreground*) sempre teriam a prioridade.

12

6 - Algoritmos Fair-share e Loteria

A política de agendamento fair-share tem como base promover a distribuição do uso do recurso de processamento de forma justa entre os diferentes atores.

Modelos como o FIFO e o SJF, por não serem estabelecidos com requisitos de prioridade, tratam todos os processos de forma igualitária. Entretanto, segundo esta lógica, em um ambiente onde poucos usuários criam muitos processos, o uso da capacidade de processamento não seria distribuído de forma igual entre os diferentes usuários.

Tomemos como exemplo uma fila de execução na qual existem 10 processos alocados com a mesma duração de processamento, sendo que cinco destes processos são de propriedade de um mesmo usuário e os outros cinco possuem diferentes donos. Baseado nesta realidade, a maioria dos algoritmos de agendamento direcionaria 50% do tempo de processamento a apenas um indivíduo, sendo a outra metade compartilhada entre os donos dos demais processos.

Para evitar esta situação, o algoritmo fair-share divide o tempo de processamento não pela quantidade de processos, mas pela quantidade de donos. Desta forma, mesmo com um mesmo usuário tendo cinco dos dez processos na fila de espera, este teria apenas cerca de 17% do tempo de processamento disponível na CPU, ou seja, o mesmo percentual dos outros cinco usuários donos de um único processo cada. O algoritmo fair-share se baseia no modelo preemptivo.

Modelo tradicional	Modelo fair-share
100% do tempo de processamento / 10 processos = 10% do tempo de processamento para cada processo	100% do tempo de processamento/ 6 usuários = 17% do tempo de processamento para cada usuário independentemente do número de processos de cada um

© 2015 - AIEC - Associação Internacional de Educação Continuada

13

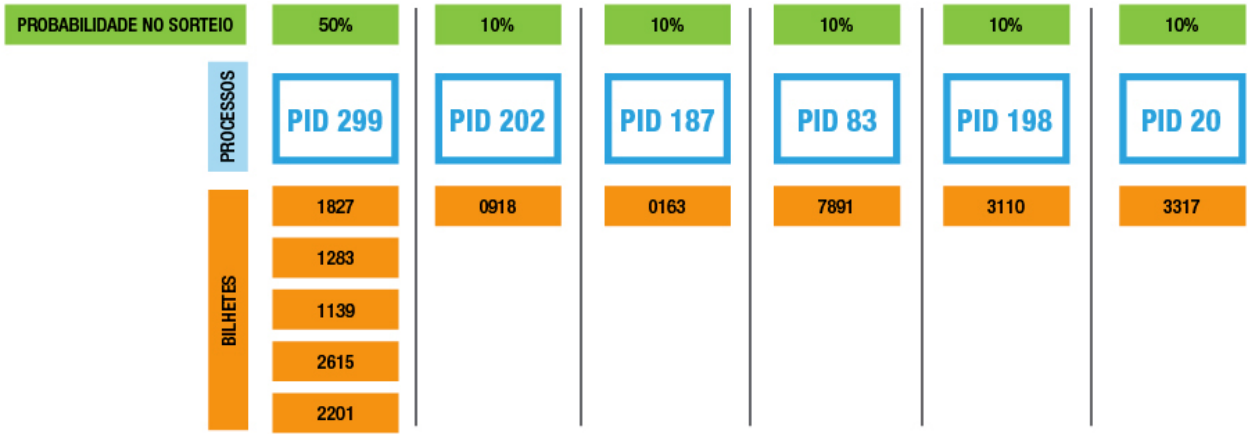
Loteria

A política de agendamento por loteria tem como fundamento a aleatoriedade na alocação dos processos para execução.

O método é implementado através da distribuição de números randômicos para cada um dos processos, como forma de se representar bilhetes de loteria. A escolha do bilhete do processo a ser posto em execução seria posta em prática através da realização de sorteio, também aleatório, entre os números distribuídos.

Esta estratégia permite duas abordagens, tanto a do tratamento igualitário quando a priorização de determinados processos. Se o sistema operacional distribui um único ticket para cada um dos *n* processos, cada um destes passaria a ter a mesma probabilidade de entrar em execução – 1/n.

Ao mesmo tempo, se há a necessidade de se priorizar determinados processos em detrimento de outros, o Sistema Operacional pode fornecer bilhetes adicionais para os processos prioritários, aumentando a sua probabilidade de ser sorteado. Um processo com cinco bilhetes, por exemplo, teria a probabilidade de 5/n de ser posto em execução, conquanto os demais continuariam mantendo a probabilidade original de 1/n. Assim como o fair-share, o algoritmo loteria também é preemptivo.



14

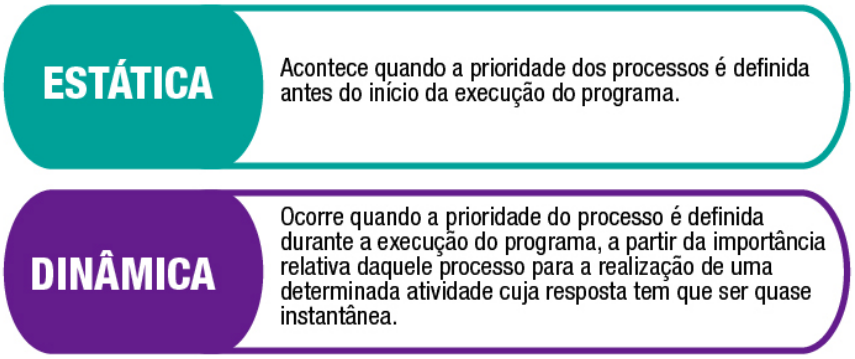
7 - Tempo Real

Os sistemas em tempo real têm uma particularidade em relação aos demais sistemas, já que esperam não apenas que a resposta do processamento esteja certa, mas que chegue dentro do tempo esperado para uma resposta imediata.

Este é o caso, por exemplo, de sistemas de monitoramento de pacientes ou de aplicativos utilizados para controle do tráfego aéreo.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Os sistemas em tempo real podem possuir política de escalonamento **estática** ou **dinâmica**.



15

Resumo

Apesar de aparentar ao usuário final que os processos estão sendo executados simultaneamente, o que ocorre na prática é que apenas um processo ocupa o processador por vez. A ilusão de simultaneidade se dá como consequência da rápida alternância de contexto de processos em um pequeno intervalo de tempo. Para realizar efetivamente esta atividade de gerenciamento de processos, o SO se baseia em uma série de algoritmos para promover o agendamento dos processos.

- **First-In-First-Out**: baseia-se na premissa de que os primeiros processos a solicitar a execução serão os primeiros a serem atendidos. Assim, é criada uma fila que ordena os processos que serão postos em execução por ordem de chegada. Já o algoritmo Shortest-Job-First tem como premissa que os processos que demandarem menor tempo de execução serão os primeiros a serem executados, independente da ordem de chegada. A principal consequência da utilização deste método de escalonamento é a redução do tempo médio de espera para execução global do conjunto de processos.
- **Round Robin**: é similar ao First-In-First-Out, já que ordena a fila de execução a partir da ordem de chegada dos processos. A diferença principal entre os dois algoritmos é que o Round Robin associa a cada processo um tempo máximo de execução a ser utilizado naquele ciclo de uso do processador, impedindo-o de monopolizar o acesso a CPU por um largo período de tempo.
- O **algoritmo por prioridades** se baseia na atribuição de um valor de prioridade a cada processo no momento de sua criação, baseado em um parâmetro específico como, por exemplo, o cargo do usuário que solicitou a execução do processo. Já o método de **múltiplas filas**, como o próprio nome já diz, cria diversas filas com diferentes prioridades que permitem agrupar os processos com prioridade similar. Desta forma, as solicitações agrupadas nas filas de menor prioridade só são executadas no caso de não existirem processos nas filas de maior prioridade.
- **Fair-share**: promove uma distribuição justa dos recursos entre os diferentes atores. Diferentemente de modelos como o First-In-First-Out, realiza a divisão do tempo disponível para processamento em função da quantidade de usuários demandantes dos processos, e não da quantidade de processos na fila de execução.
- **Escalonamento em loteria** tem como fundamento a distribuição de números aleatórios para cada processo e a alocação destes a partir da realização de um sorteio, também randômico. Este

© 2015 - AIEC - Associação Internacional de Educação Continuada

modelo permite a aplicação de prioridade a processos a partir da distribuição de números extras para o sorteio.

- **Tempo real**: diferentemente dos sistemas convencionais, para que a operação seja executada com sucesso, não basta que o resultado retornado esteja correto, mas também que a resposta seja devolvida dentro do tempo esperado. Esta categoria de sistemas pode possuir política de escalonamento estática ou dinâmica, a depender da sua aplicação.

UNIDADE 2 – PROCESSOS E MEMÓRIA

MÓDULO 4 – CONCEITOS SOBRE O GERENCIAMENTO DE MEMÓRIA, PAGINAÇÃO DA MEMÓRIA

01

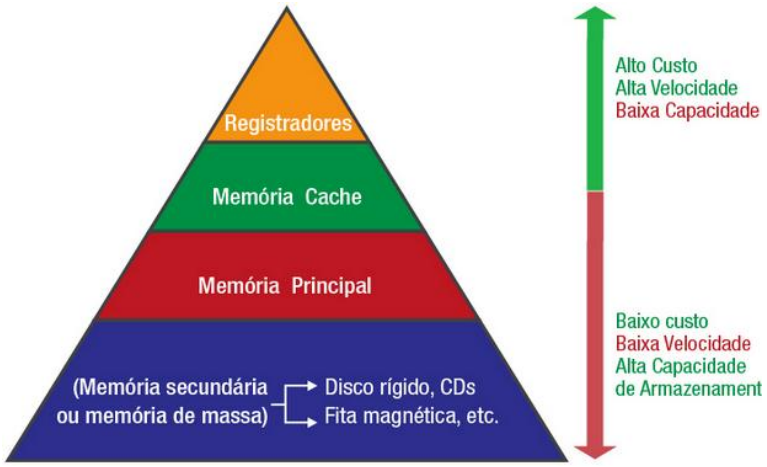
1 - GERENCIAMENTO DE MEMÓRIA

Apesar de muitas vezes parecer para os programadores de *software* que a quantidade de memória RAM do computador é praticamente ilimitada e é suficiente para prover os recursos necessários para a execução simultânea de dezenas de aplicativos, esta não é a realidade encontrada na maioria dos computadores.

Apesar da redução de preço das memórias de rápido acesso observada nas últimas décadas, a exemplo da memória cache presente nos atuais processadores e da própria memória RAM utilizada nos computadores, o custo de se ter grandes quantidades de memórias com altas taxas de acesso ainda é impeditivo para a maioria dos usuários de computador.

Por conta desta realidade, os computadores possuem diferentes tipos de memória, com velocidades e custos distintos, conceito que é conhecido como hierarquia de memória. Toda a tarefa de gerenciar o uso destas diferentes memórias é realizada pelo sistema operacional.

Como já vimos, os computadores são fornecidos com memórias que possuem poucos megabytes de armazenamento de altíssima velocidade de acesso – como a memória cache, com memórias de alta velocidade e que possuem capacidade de armazenamento de gigabytes – a exemplo da memória RAM, e discos de média velocidade, mas com capacidade de armazenamento de terabytes – como os discos rígidos.



© 2015 - AIEC - Associação Internacional de Educação Continuada

É interessante ressaltar que o custo destes meios de armazenamento diminui na mesma direção que a velocidade de acesso aos dados provida.

02

Cada um dos tipos de memória gerenciados pelo sistema operacional tem uma característica própria:

- O **espaço em disco**, mais barato e em maior quantidade, é normalmente utilizado para armazenar arquivos de usuário e de programas que não estão em execução.
- A **memória principal** armazena dados de processos e aplicativos que estão atualmente em execução e é um meio de armazenamento volátil.
- A **memória cache** é fornecida em conjunto com o processador e tem como principal função acelerar a velocidade de processamento.
- Os **registradores** são os meios de armazenamento de maior custo e de maior velocidade de acesso, são diretamente ligados a CPU e são vinculados, dentre outras tarefas, a dar suporte às instruções que estão em execução em um dado momento.

Um dos principais objetivos do gerenciamento de memória é justamente o de otimizar o fluxo dos arquivos entre os diferentes meios de armazenamento de modo a impedir que a baixa velocidade de acesso de alguns elementos da hierarquia de memória possam influenciar negativamente na velocidade de execução dos aplicativos.

Além disso, como o processador só é capaz de executar instruções que estão armazenadas na memória principal, quando um determinado aplicativo é posto em execução, tem seus arquivos obrigatoriamente transferidos da memória secundária para a principal.

03

- **Endereçamento Físico e Lógico**

Os primeiros sistemas operacionais utilizavam um esquema de alocação de memória conhecido como contíguo.

No modelo **contíguo**, a memória era dividida em duas partições, uma onde se mantém residente o próprio sistema operacional e outra para prover armazenamento para os processos em execução.

Além disso, os aplicativos que rodavam sobre o SO utilizavam apontamentos diretos para o endereço físico da memória, ou seja, era fornecido acesso a localização exata do armazenamento provido pela unidade de memória principal.

Entretanto, este esquema de gerenciamento de memória apresentava algumas desvantagens, sobretudo porque a utilização do endereço físico permitia a ocorrência de uma série de problemas, como a possibilidade, em ambientes multiprogramáveis, de determinados processos acessarem a área de memória alocada para outros. Estas deficiências foram um dos fatores que motivaram o surgimento dos esquemas de endereçamento lógico de memória.

© 2015 - AIEC - Associação Internacional de Educação Continuada

O **endereçamento lógico** é um identificador gerado pela CPU que direciona para um endereço físico na memória, impedindo aos processos o acesso direto a este componente de *hardware*.

A utilização do endereçamento lógico foi o que permitiu, na prática, a criação dos espaços de endereçamento de cada processo, criando áreas de memória de acesso exclusivo.

A unidade de gerenciamento de memória (MMU) é o dispositivo de *hardware* responsável por realizar o mapeamento entre os endereços lógico e físico.

04

- **Memória Virtual**

O conceito de memória virtual tem como um dos pilares a utilização do esquema de endereçamento físico e lógico e surgiu tendo como um dos objetivos sanar o problema do espaço insuficiente de armazenamento disponível em memória principal.

A ideia básica por trás da **memória virtual** é a atribuição de espaços de endereçamento lógico exclusivos para cada processo, sendo que este espaço é dividido em páginas.

Estas páginas, apesar de mapeadas na memória principal, nem sempre estão fisicamente nesta hierarquia de memória durante a execução do programa.

Os espaços de endereçamento lógicos, ou virtuais, permitem que o processador faça referência a pontos de armazenamento físicos localizados também na memória secundária. Assim, quando um processo precisa realizar o acesso a localização física, é necessário apenas efetuar o mapeamento entre os endereços.

Na prática, utilização da memória virtual cria a ilusão de que a memória secundária é uma extensão da memória principal, já que parte do programa é mantida em memória secundária e apenas quando alguma instrução da página é referenciada, está é carregada na memória principal.

05

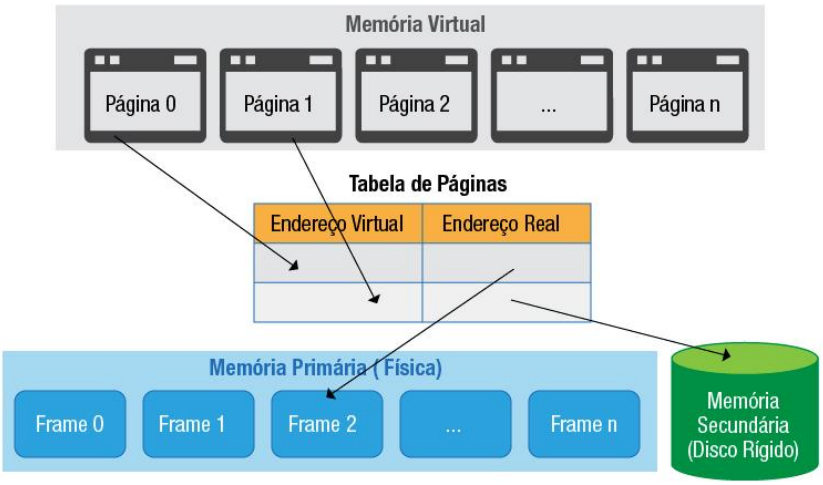
2 - Algoritmos de Paginação de Memória

A maior parte dos sistemas operacionais que implementam o método de memória virtual utilizam a paginação.

De acordo com a técnica de **paginação**, tanto o espaço de endereçamento virtual quanto o real são divididos em estruturas de mesmo tamanho chamadas, respectivamente, de **páginas virtuais** e ***frames***. Estas estruturas se relacionam através de uma tabela de páginas que contém tanto o endereço virtual quanto o endereço físico.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Uma das partes mais importantes da paginação é o fato de que ela permite que existam *frames* tanto na memória primária quanto na secundária, o que viabiliza a execução de aplicativos maiores do que a quantidade de memória primária disponível.



06

Carregar dados em memória secundária, entretanto, não é uma solução definitiva, já que o código do aplicativo que está em execução naquele dado momento precisa obrigatoriamente estar em memória primária para que possa ser acessado pelo processador.

Assim, sempre que um *frame* referenciado a partir de uma instrução executada pelo processador não é encontrado na memória principal, ocorre o que é conhecido como **falha de página** (*page fault*). Esta falha inicia no sistema operacional o processo automático de transferência da página solicitada da memória secundária para a memória primária.

Caso não exista espaço disponível na memória primária no momento da transferência, é necessário remover *frames* ociosos com o objetivo de liberar espaço de armazenamento, o que motivou a criação de uma série de algoritmos voltados exclusivamente a propor métodos de escolha das páginas a serem substituídas.

Neste módulo serão estudados sete **algoritmos voltados à substituição de páginas em memória**:

- FIFO,
- Ótimo,
- página Menos frequentemente utilizada,
- Menos recentemente utilizada e
- Não recentemente utilizada,
- Segunda Chance, e
- relógio.

07

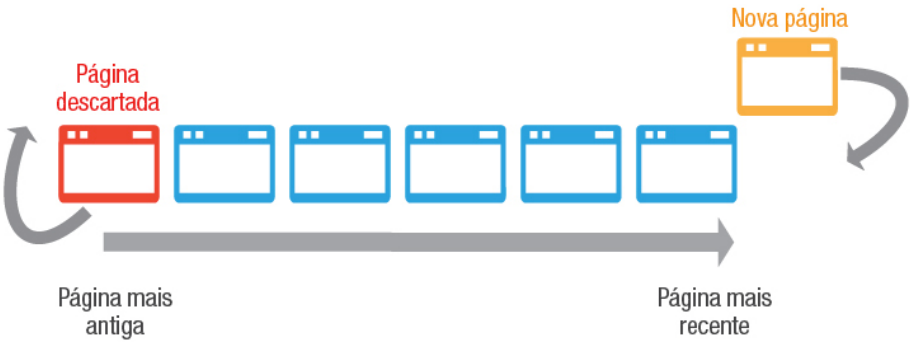
a) First In First Out - FIFO

© 2015 - AIEC - Associação Internacional de Educação Continuada

Este algoritmo segue a mesma lógica apresentada na política de agendamento do processador apresentada em módulo anterior.

A ideia do algoritmo **FIFO** é a de que as páginas são organizadas em uma espécie de fila interligada, onde a primeira página da fila, e consequentemente a mais antiga, é a primeira a ser descartada.

O método parece óbvio, já que é trivial se pensar que o pacote armazenado mais antigo tem grande possibilidade de ser menos útil do que os pacotes mais recentes, entretanto, o FIFO não leva em consideração a quantidade de vezes em que o bloco é referenciado desde que foi alocado, o que faz com que blocos com grande quantidade de acessos recentes sejam removidos pelo simples fato de serem os mais antigos.



08

b) Ótimo

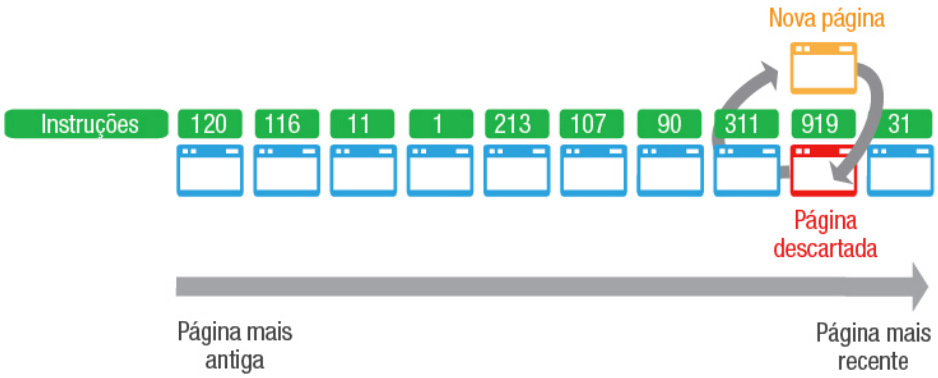
Esse algoritmo representa a resolução ideal para o problema de substituição de páginas pelo sistema operacional.

A ideia do algoritmo **ótimo** é a de que cada página alocada em memória tenha uma variável associada que indique a quantidade de instruções que seriam executadas até que cada uma das páginas fosse novamente referenciada.

Desta forma, as páginas que tivessem que aguardar a maior quantidade de instruções antes de serem acessadas seriam as primeiras a serem substituídas.

O grande problema deste algoritmo é que ele não é implementável, ou seja, não é possível de ser codificado, já que não há como prevê a quantidade de instruções que antecederiam a chamada de uma determinada página.

© 2015 - AIEC - Associação Internacional de Educação Continuada



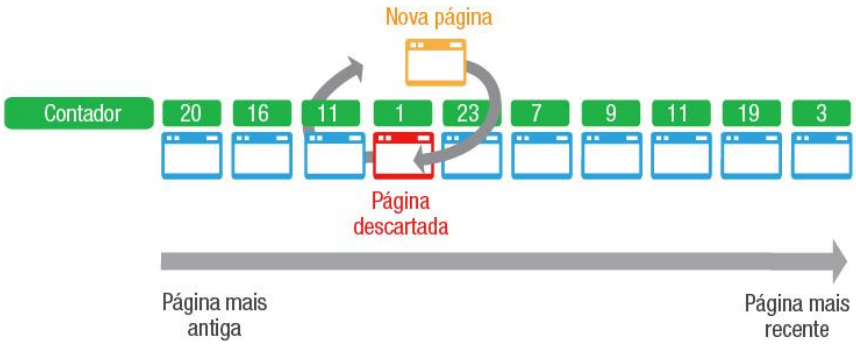
09

c) Menos frequentemente utilizada (Least Frequently Used)

A estratégia definida por este algoritmo é a utilização de um contador de acessos para cada uma das páginas em memória. Assim, o contador de cada página seria incrementado a partir de um novo acesso específico a esta página, sendo que em caso de necessidade de substituição de uma das páginas, seria removida aquela cujo contador apresentasse o menor número de acessos.

A lógica é simples, mas nem sempre funciona, já que este método não tem vinculação com a política de substituição FIFO, ou seja, não leva em consideração o momento em que uma determinada página foi alocada em memória.

Isto faz com que páginas que foram recentemente carregadas e, por conseguinte, ainda tem poucos acessos, sejam substituídas em detrimento de páginas antigas que têm centenas de acessos marcados no contador, mas que não foram referenciadas recentemente.



10

d) Não recentemente utilizada (Not recently used)

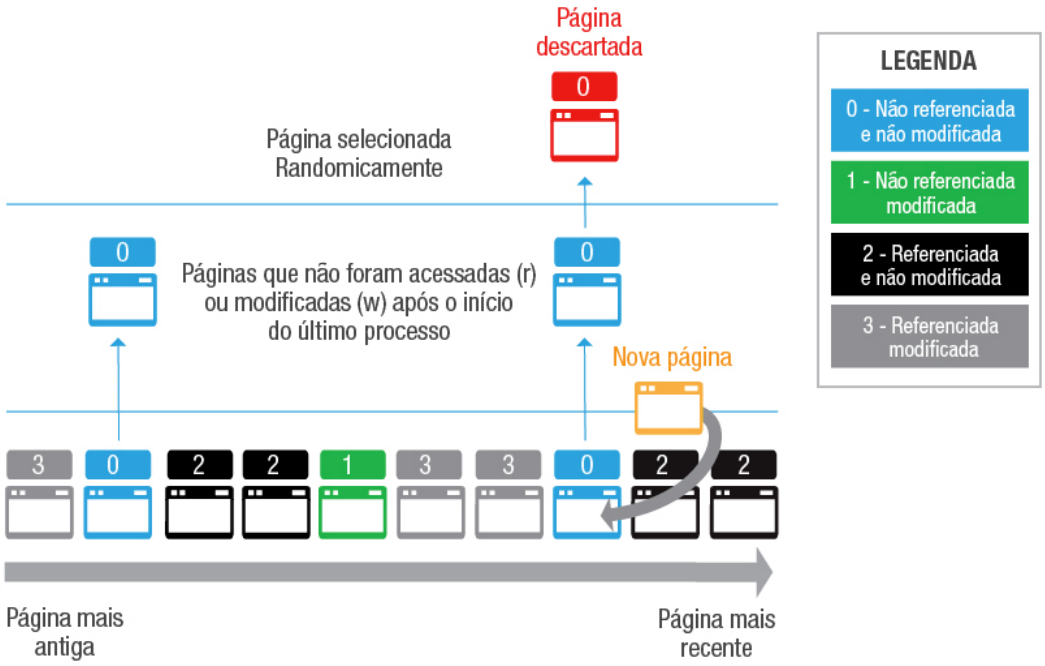
O algoritmo se substituição da página **não recentemente utilizada** tem base na análise de estatísticas das páginas relacionadas a quando estas foram referenciadas ou modificadas.

© 2015 - AIEC - Associação Internacional de Educação Continuada

Cada página tem marcadores que são utilizados especificamente para indicar o momento em que foram referenciadas ou modificadas, sendo que quando um novo processo é iniciado os marcadores de todas as páginas são zerados.

Com base nestes marcadores, o sistema operacional classifica as páginas alocadas em quatro categorias:

- Classe 0 – páginas que não foram referenciadas nem modificadas.
- Classe 1 - páginas que não foram referenciadas, mas foram modificadas.
- Classe 2 - páginas que foram referenciadas, mas não foram modificadas.
- Classe 3 - páginas que foram referenciadas e modificadas.



Desta forma, quando é necessário substituir uma página da memória, o sistema operacional verifica o conjunto de páginas alocadas na classe de menor número, neste caso Classe 0, e seleciona randomicamente a que será removida para dar lugar a nova página.

11

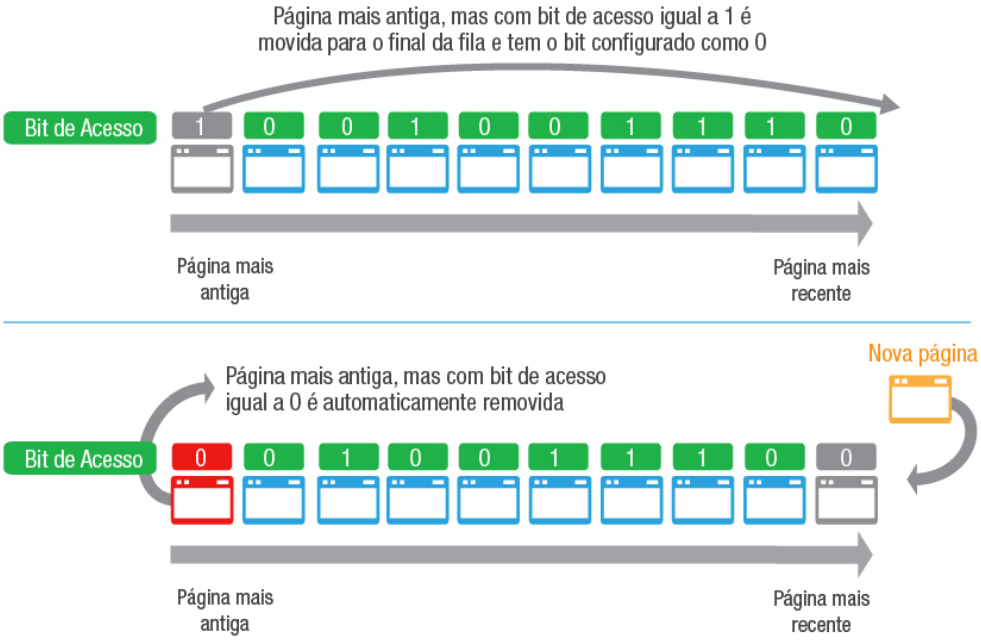
e) Segunda Chance

Este algoritmo utiliza parte da lógica implementada no FIFO, só que evitando o problema de se remover uma página que, apesar de antiga, ainda é muito referenciada. Para isso, o método leva em consideração não apenas o momento de alocação da página em memória, mas observa se o bit que indica se a página foi recentemente acessada está assinalado.

Na prática, se a página for a mais antiga e o bit que indica acesso recente estiver marcado como 0, a página é automaticamente removida para dar acesso a nova página. Nos casos onde a página mesmo sendo a mais antiga tem o bit de acesso recente marcado como 1, esta é automaticamente colocada

© 2015 - AIEC - Associação Internacional de Educação Continuada

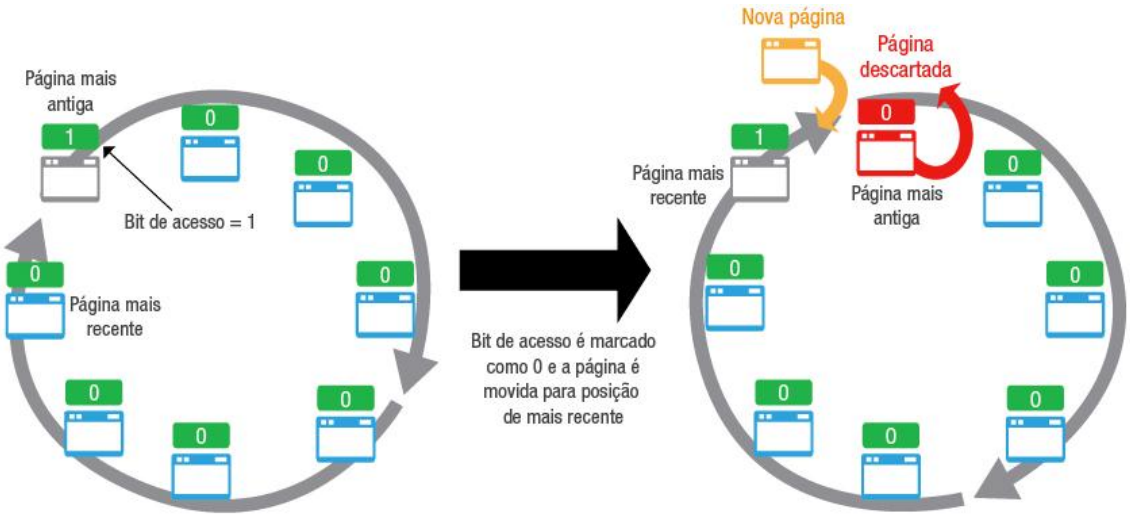
para o final da fila e tem o bit de acesso configurado como 0, sendo que a análise para substituição continua com a próxima página mais antiga.



12

f) Relógio

A lógica deste método é baseada na do algoritmo de segunda chance, com a diferença de que as páginas são organizadas em uma lista circular. Esta pequena modificação evita a movimentação da página mais antiga, mas que foi recentemente modificada para o final da lista, já que, ao invés disso, basta movimentar a posição do ponteiro na lista circular.



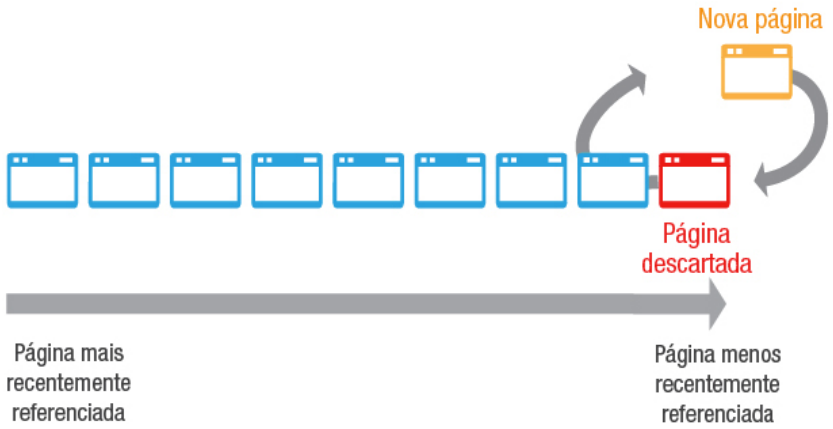
© 2015 - AIEC - Associação Internacional de Educação Continuada

13

g) Menos recentemente utilizada (Least recently used)

O algoritmo em questão tem como critério de substituição da página a seleção daquela que está a mais tempo sem ser utilizada. Assim, não é levado em consideração nem a sequência de alocação das páginas nem a quantidade de vezes que foi utilizada, apenas o momento do último acesso é utilizado como critério de seleção da página a ser descartada. Por conseguinte, as páginas são normalmente ordenadas em uma lista a partir do momento do último acesso, e não do momento em que foram alocadas em memória.

O ponto negativo deste método é que sempre que uma nova instrução faz referência a uma determinada página em memória a lista deve ser atualizada e a página recentemente acessada deve ser movida para o início da fila. Na lógica deste algoritmo, a página a ser removida é sempre a última da lista.



14

RESUMO

O processo de gerenciamento de memória é uma das principais tarefas desempenhadas pelo sistema operacional. Os grandes desafios desta atividade é permitir que múltiplos processos consigam compartilhar uma mesma estrutura de memória principal, além de fazer que aplicativos maiores do que a capacidade de memória primária instalada consigam ser executados com sucesso.

Uma das principais soluções utilizada para conseguir que múltiplos processos tenham a capacidade de compartilhar a mesma estrutura de memória é a utilização do endereçamento lógico, associado ao físico. Desta forma, é possível alocar para cada processo um conjunto único e exclusivo de endereços, evitando problemas no acesso aos recursos de armazenamento na memória principal.

Já a memória virtual está relacionada à execução de programas maiores do que o espaço disponível na memória principal. A utilização deste dispositivo faz com que os elementos de memória secundária,

© 2015 - AIEC - Associação Internacional de Educação Continuada

como os discos rígidos, sejam vistos como uma extensão da memória RAM, aumentando virtualmente a capacidade de armazenamento em memória principal.

A maior parte dos sistemas operacionais modernos implementam o método de memória virtual através do uso da técnica de paginação, que funciona através da divisão do espaço de endereçamento em estruturas de mesmo tamanho chamadas de páginas. A paginação cria as estruturas de suporte tanto no endereçamento real – onde são chamadas de *frames*, quando no endereçamento virtual – onde são denominadas páginas virtuais.

Apesar da ilusão de uma memória única, na prática, a parte específica do código do programa que está em execução em um dado momento ainda precisa estar alocado em memória principal. Se no momento em que for referenciado pelo processador o endereço virtual estiver apontando para uma área na memória secundária, esta página deve obrigatoriamente ser transferida para a memória principal.

Como muitas vezes o programa é alocado em memória secundária por falta de espaço na principal, para que uma página seja transferida é necessário primeiro liberar espaço de armazenamento. Para implementar este processo de substituição de páginas foram propostos diversos algoritmos, sendo os mais conhecidos os algoritmos do relógio, o FIFO, o Ótimo, o da página menos frequentemente utilizada, o da página não recentemente utilizada, o da segunda chance e o da página menos recentemente utilizada.

© 2015 - AIEC - Associação Internacional de Educação Continuada