

Multiprogramação

Em um **sistema multiprogramado** diversos programas são mantidos na memória ao mesmo tempo. Quando temos, por exemplo, 3 programas para serem executados na memória principal, o programa 1 faz a chamada de um periférico e enquanto o periférico executa o comando enviado, o SO inicia a execução por exemplo, do programa 2. **Assim, processador e periférico trabalham ao mesmo tempo.** A maioria dos programas não precisa de toda memória do computador. Na verdade, a maioria deles ocupa apenas uma parcela da memória principal disponível. **Sem a multiprogramação a memória não ocupada por um programa ficaria sem utilização.** Com vários programas na memória, esse recurso também é melhor aproveitado.

Conceito de processo

Na maioria das vezes, um processo é definido como *"um programa em execução"*. O conceito de processo é bastante abstrato, mas essencial ao estudo de sistemas operacionais.

Um programa é uma seqüência de instruções. É algo passivo dentro do sistema. Ele não altera o seu próprio estado. **O processo é um elemento ativo.** O processo altera seu estado, à medida que executa um programa. **É o processo que faz chamadas de sistema, ao executar os programas.**

É possível que **vários processos executem o mesmo programa ao mesmo tempo.** Por exemplo, diversos usuários podem estar utilizando simultaneamente o mesmo editor de texto. Existe apenas um programa 'editor de texto', porém, para cada usuário existe um processo executando o programa. Cada processo representa uma execução independente do editor de textos. Todos os processos utilizam uma mesma cópia do código do editor de textos, porém cada processo trabalha sobre uma área de variáveis privativa.

Ciclos de um processo

Processos são criados e destruídos. Alguns sistemas trabalham com um número fixo de processos. Nesse caso, todos os processos são criados na inicialização do sistema e só são destruídos quando o sistema é desligado.

A maioria dos processos do sistema executam programas dos usuários. Entretanto, **alguns podem realizar tarefas do sistema.** São processos do sistema, não de usuários(daemon). Um exemplo clássico é o spooling da impressora, onde um processo do sistema copia os arquivos desse diretório para a impressora. Isso é feito para evitar conflitos na utilização da impressora e também para evitar que os usuários precisem ficar "atentos" à espera da desocupação da impressora.

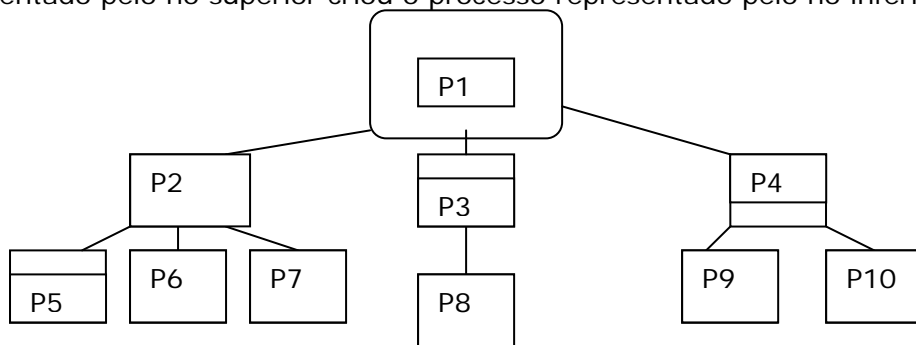
Após ter sido criado, o processo passa a ocupar o processador. Em determinados momentos, ele deixa de ocupar o processador para realização de uma operação de I/O. **Os momentos em que um processo não está esperando por I/O, ou seja, em que ele deseja ocupar o processador, são chamados de "ciclos de processador"**. Já quando o processo está esperando por uma operação de I/O, ele está em um ciclo de I/O. A conclusão da chamada de sistema faz o caminho inverso. **O primeiro ciclo da vida de um processo será necessariamente um ciclo de processador.** Para entrar em um ciclo de I/O é necessário executar ao menos uma instrução. No caso, a instrução que faz a chamada do sistema.

Um processo que utiliza muito o **processador** é chamado de **cpu-bound**. O seu tempo de execução é definido principalmente pelo tempo dos seus ciclos de processador. Por outro lado, um processo que utiliza muito **I/O** é chamado **I/O bound (input/output bound)**. Nesse caso, o tempo de execução é definido principalmente pela duração das operações de I/O. Um processo que faz a cópia de um arquivo é I/O bound, porque praticamente não ocupa o processador. Já um cálculo complexo utiliza apenas o processador, logo é cpu-bound. Exemplo: um CTRL+C > CTRL+V é I/O bound.

A renderização de um vídeo é cpu-bound. O ideal é ter no sistema uma mistura de processos cpu-bound e I/O bound.

Relacionamento entre processos

Alguns sistemas suportam o conceito de **grupos de processos**. Se processos são criados através de chamada de sistema, **pode-se considerar como pertencentes ao mesmo grupo todos os processos criados pelo mesmo processo**. O conceito de grupo permite que operações possam ser aplicadas sobre todo um conjunto de processos, e não apenas sobre procesos individuais. Por exemplo, os processos pertencentes a um mesmo grupo podem compartilhar os mesmo direitos perante o sistema. **Em muitos sistemas os processos são criados por outros processos, através de chamada de sistema. Nesse caso, é possível definir uma hierarquia de processos**. O processo que faz a chamada de sistema é chamado de **processo pai**. O processo criado é chamado de **processo filho**. Um mesmo processo pai pode estar associado à vários processos filhos. Os processos filhos, por sua vez, podem criar outros processos. Essa situação é facilmente representada através de uma árvore. Cada nó da árvore representa um processo. Uma ligação entre dois nós significa que o processo representado pelo nó superior criou o processo representado pelo nó inferior.



É importante ressaltar que o formato da árvore muda com o passar do tempo. À medida que processos são criados e destruídos, nós são acrescentados ou removidos. O sistema também pode definir, por exemplo, **o que fazer quando um nó pai é destruído. Os nós filhos deverão sobrar ou não?**

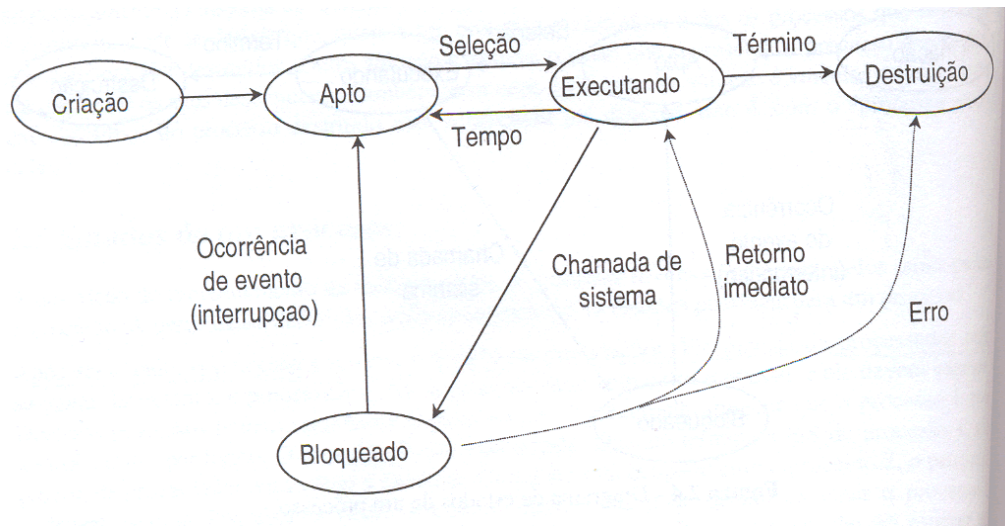
Estados de um processo

Após ser criado, o processo entra em um ciclo de processador. Ele precisa de processador para executar. Entretanto, **o processador poderá estar ocupado com outro processo, e ele deverá esperar**. Diversos processos podem estar no mesmo estado.

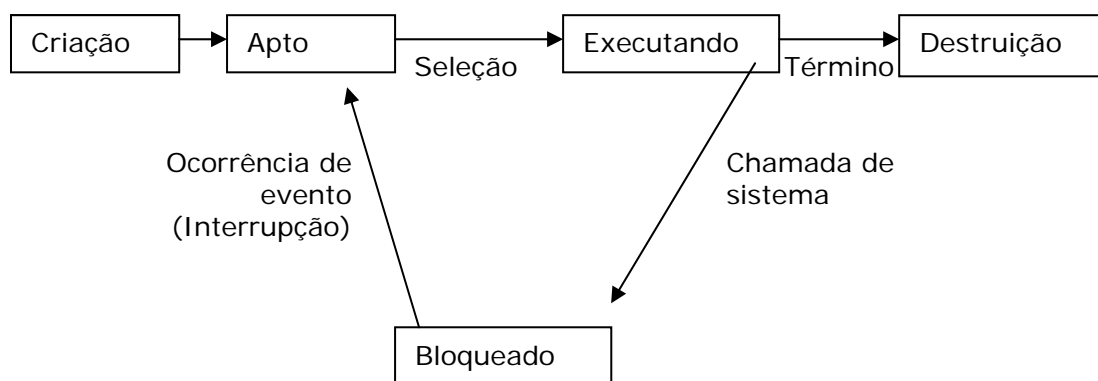
Em máquinas multiprocessadoras existem diversos processadores. Nesse caso, diversos processos executam ao mesmo tempo. Porém, essa não é a situação mais comum. Vamos supor que existe um único processador no computador. Nesse caso, é necessário manter uma fila com os processos aptos a ganhar o processador. Essa fila é chamada **"fila de aptos" (ready queue)**.

Processador





Existe também o diagrama de estados de um processo. No estado executando, um processo pode fazer chamadas de sistema. **Até a chamada de sistema ser atendida, o processo não pode continuar sua execução.** Ele fica bloqueado e só volta a disputar o processador depois da conclusão da chamada. Enquanto espera pelo término da chamada de sistema, o processo está no estado bloqueado (blocked).



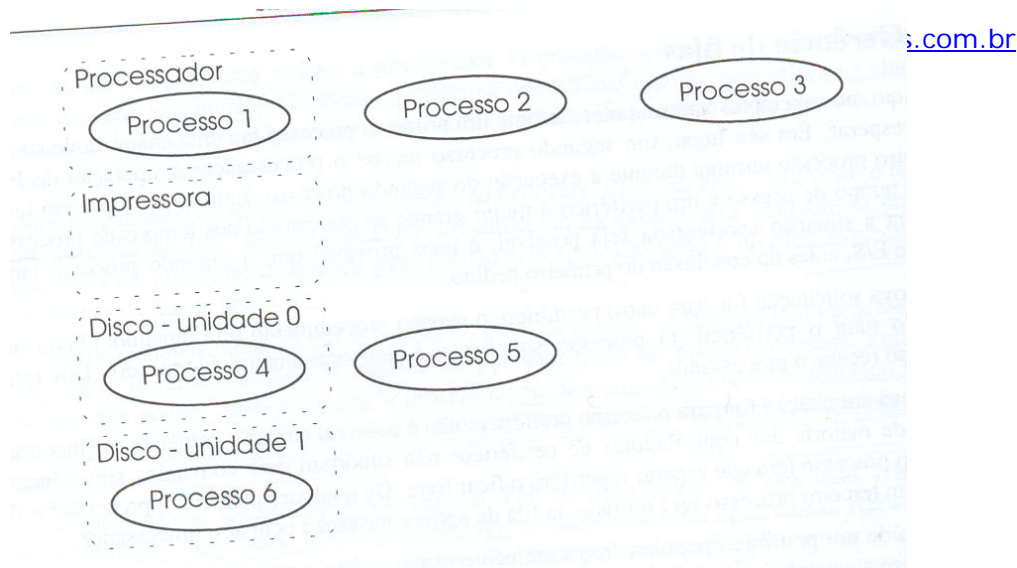
A mudança de estado de qualquer processo é iniciada por um evento. Esse evento aciona o sistema operacional, que então altera o estado de um ou mais processos. A transição do estado executando para o bloqueado é feita através de uma chamada de sistema. Uma chamada de sistema é necessariamente feita pelo processo no estado executando. Ele fica no estado bloqueado até o atendimento. Com isso, o processador fica livre. O Sistema Operacional então seleciona um processo da fila de aptos para receber o processador. O processo selecionado passa do estado se apto para o estado executando. **O módulo do SO que faz essa seleção é chamado escalonador (scheduler).**

O processo liberado passa do estado de bloqueado para o estado de apto. Ele volta a disputar o processador com os demais da fila de aptos.

Alguns outros caminhos são possíveis no grafo de estados: A destruição do processo pode ser em função de uma chamada de sistema ou por solicitação do próprio processo. **Entretanto alguns sistemas podem resolver abortar o processo em caso de erro crítico no processo de I/O.** Nesse caso, passa a existir um caminho do estado bloqueado para a destruição.

Muitos sistemas procuram evitar que um único processo monopolize a ocupação do processador. **Se um processo está há muito tempo no processador, ele volta para o fim da fila de aptos.** Um novo processo da fila de aptos ganha o processador.

Gerência de filas



É preciso lembrar que o tempo de acesso a um periférico é muito grande se comparado aos tempos de um processador, assim sendo, na maioria das vezes, um segundo processo também solicite I/O, antes da conclusão do mesmo pedido.

Se a nova solicitação for para outro periférico, o mesmo procedimento será repetido. O comando é enviado para o periférico. **Assim, o processo solicitante é temporariamente bloqueado e um terceiro processo recebe o processador.**

Se a nova solicitação for para o mesmo periférico, não é possível enviar o comando imediatamente. A grande maioria dos controladores de periféricos não suportam dois comandos simultâneos. **O segundo processo deverá esperar o periférico ficar livre.** de qualquer maneira, o processador ficou livre. Um terceiro processo será retirado da fila de aptos e passará a ocupar o processador.

No caso de um periférico frequentemente usado (como o HD) essa mesma situação pode ocorrer várias vezes.

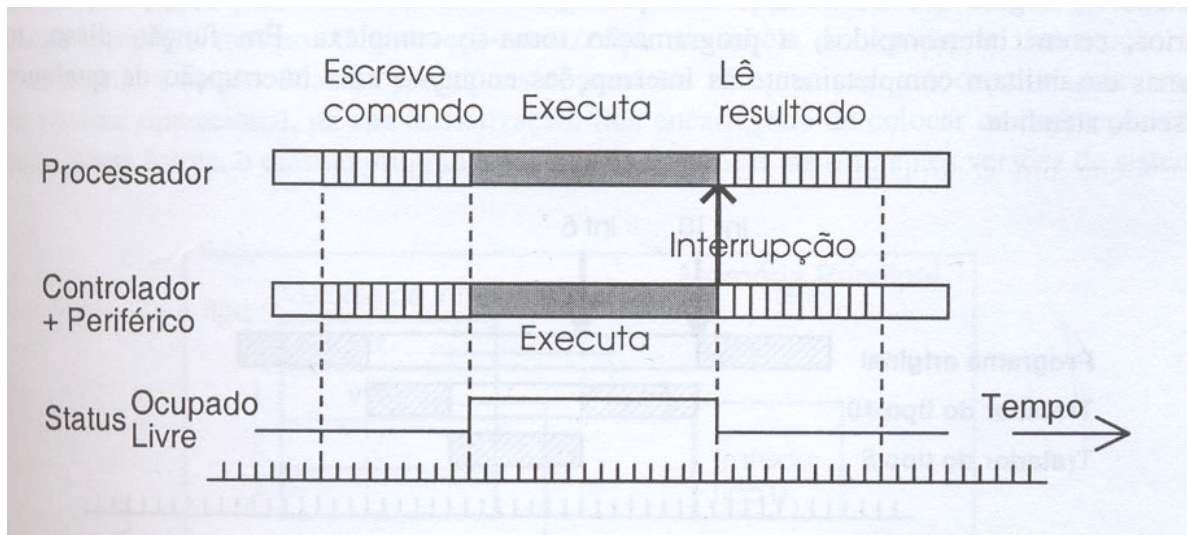
Quando uma solicitação é feita, é preciso verificar a fila do periférico. Se a fila estiver vazia, o periférico está livre. O pedido é inserido na fila, e o comando é enviado ao controlador. Caso contrário, o periférico está ocupado. Na interrupção do periférico também são necessários alguns cuidados adicionais. **O primeiro pedido da fila pode ser removido, pois o acesso foi concluído. O processo correspondente volta para a fila dos aptos, para disputar o processador. Se, após sua remoção, a fila ficou vazia, então o periférico está livre.** Caso contrário, é necessário enviar para o controlador do periférico o primeiro pedido da fila. Trata-se de uma solicitação feita anteriormente, mas que não fora enviada pois o periférico estava ocupado.

Mecanismo de Interrupções

O mecanismo de interrupções é um recurso comum dos processadores de qualquer porte. Ele permite que um **controlador de periférico chame a atenção do processador.** Fisicamente, o barramento de controle é usado para o envio de sinais elétricos associados com a geração de uma interrupção.

A analogia é entre hardware e telefone: Imagine uma pessoa trabalhando (o processador) quando toca o telefone (ocorrência de interrupção). A pessoa pára o que está fazendo e vai atender o telefone (a interrupção). Quando o atendimento tiver terminado, a pessoa(processador) volta para o que estava fazendo, no ponto em que parou.

Uma interrupção sempre sinaliza a ocorrência de um evento. Quando ela acontece, desvia a execução da posição atual de programa para uma rotina específica. **Essa rotina**, responsável por atender a interrupção, é chamada de **tratador de**



interrupção. O tratador realiza as ações necessárias em função da ocorrência da interrupção. Ele é, simplesmente, uma rotina que somente é executada quando ocorre uma interrupção. Quando o tratador termina, a execução volta para a rotina interrompida, sem que essa perceba que foi interrompida.

Para que a execução do programa interrompido não seja comprometida pela interrupção, é necessário que nenhum registrador seja alterado. Em outras palavras, quando a execução voltar para o programa interrompido, o conteúdo de todos os registradores deverá ser o mesmo que no momento em que ocorreu a interrupção.

É usual o processador dispor de uma instrução tipo retorno de interrupção. Essa instrução repõe o conteúdo original dos registradores e faz o processador retomar a execução do programa interrompido.

Todo computador possui uma variada gama de periféricos. A função básica de um controlador de periférico é conectar o dispositivo em questão com o processador.

Em um computador podem existir diversos controladores capazes de gerar interrupções. A forma mais simples de identificar a origem de uma interrupção é associar a cada controlador um tipo diferente de interrupção. Dessa forma, o controlador não somente interrompe o processador mas também informa qual o tipo de interrupção.

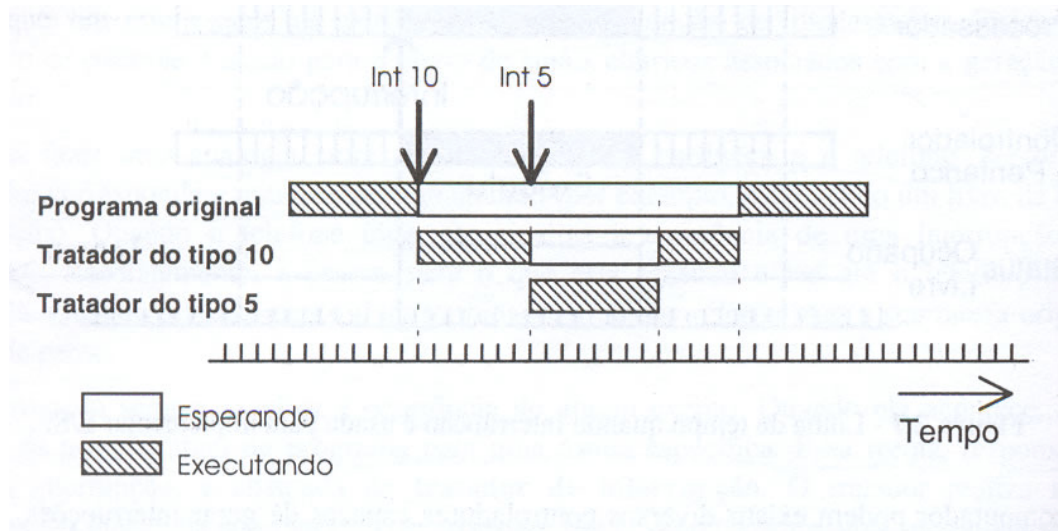
A maioria dos processadores admitem diversos tipos de interrupções. Cada tipo de interrupção é identificado por um número. Por exemplo, de 0 até 255. O significado específico de cada tipo é definido pelos projetistas do sistema.

Existem, porém, momentos em que um programa não pode ser interrompido. Por exemplo, o programa pode estar alterando variáveis que também são acessadas pelo tratador de interrupções.

É preciso ter em mente que o instante exato em que vai ocorrer uma interrupção de hardware é imprevisível.

A solução para este problema é desligar o mecanismo de interrupções temporariamente, enquanto o programa realiza uma tarefa crítica que não pode ser interrompida. Os processadores normalmente possuem instruções para habilitar e desabilitar interrupções. **Enquanto as interrupções estiverem desabilitadas, elas serão ignoradas pelo processador.** Elas não são perdidas, apenas ficam pendentes. Quando o programa tornar a habilitar as interrupções, elas imediatamente serão atendidas pelo processador. Com as interrupções desabilitadas, o acesso às estruturas de dados pode ser feito de forma segura.

É comum a existência de uma **relação de prioridades entre os diferentes tipos de interrupções.** vamos supor que um dado processador atribui prioridade de 0 até 255. Quando ocorre uma interrupção de nível 10, o processador desabilita as interrupções de 11 até 255, até que a 10 tenha sido concluída.



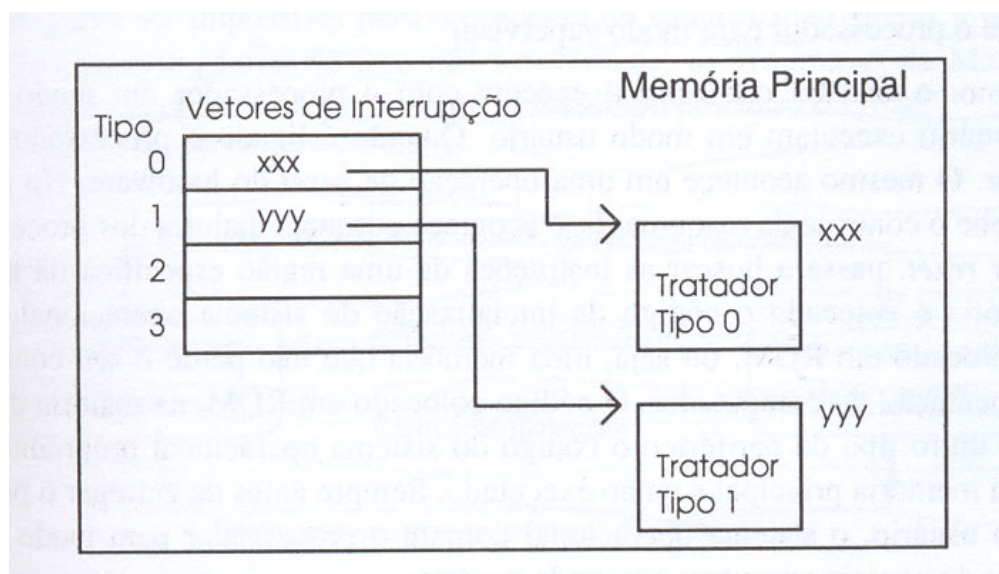
O endereço de um tratador de interrupção é chamado de **vetor de interrupção**. O termo é usado porque ele "aponta" para a rotina de atendimento da interrupção. Para ilustrar o conceito, temos a tabela dos vetores de interrupção. Para que uma mesma rotina atenda diversos tipos de interrupção, basta colocar o seu endereço em diversas entradas da tabela. Entretanto, o mais comum é utilizar uma rotina para cada tipo de interrupção.

A ocorrência de uma interrupção dispara no processador uma seqüência de atendimento. O processador verifica se o tipo de interrupção sinalizado está habilitado. Caso negativo, é ignorado. Se estiver habilitado, o conteúdo dos registradores é salvo na pilha.

Interrupções de software (também chamadas traps) são causadas pela execução de uma instrução específica para isso. Ela tem como parâmetro o número da interrupção que deve ser ativada.

O maior uso para interrupções de software é a implementação das chamadas de sistema, através das quais os programas de usuários solicitam serviços ao sistema operacional.

Não é possível desabilitar interrupções de software, mesmo porque não é necessário. Somente quem pode gerar uma interrupção de software é a rotina em execução. Se a rotina em execução não deseja que interrupções de software ocorram, basta não gerar nenhuma.



rodrigoscama@yahoo.com.br – <http://scama.norbis.com.br>

Existe uma terceira classe de interrupções geradas pelo próprio processador. **São as interrupções por erro**, muitas vezes chamadas de interrupções de exceção. Ela acontecem quando **o processador detecta algum tipo de erro na execução do programa**. O procedimento para atendimento a essa classe de interrupções é igual ao descrito anteriormente.

Proteção entre processos

Na multiprogramação diversos processos compartilham o computador. É necessário que o sistema operacional **ofereça proteção aos processos e garanta utilização correta do sistema**. Por exemplo, um usuário não pode formatar o disco.

Modos de operação do processador

A proteção apropriada implementada é usualmente definir dois **modos de operação** para o processador. Pode-se chamá-los de **modo de usuário e modo supervisor**. Quando o processador está em modo supervisor, não existem restrições, e qualquer instrução pode ser executada. Em modo usuário, algumas instruções não podem ser executadas. Essas instruções são chamadas de instruções privilegiadas, e somente podem ser executadas em modo supervisor.

Nesse mecanismo, o sistema operacional executa com o processador em modo supervisor. Os processos de usuário executam em modo usuário. Quando é ligado o processador, ele inicia em modo supervisor. O mesmo acontece ao fazer o reset da máquina. No reset, o sistema operacional recebe o controle da máquina. Isso acontece porque a maioria dos processadores, ao ser ligado ou ao sofrer reset, passa a buscar as instruções de uma região específica da memória. Nessa região da memória é colocado o código de inicialização do sistema operacional. Esse código é geralmente colocado na **ROM**, ou seja, uma memória que não perde seu conteúdo quando é desligada a alimentação do computador. O código colocado na ROM, na maioria das vezes busca no disco ou em outro tipo de periférico o código do sistema operacional propriamente dito. Ele é carregado para a memória principal e então executado.

Proteção dos periféricos

Para proteger os periféricos, as instruções I/O são tornadas privilegiadas. **Se um processo de usuário tentar acessar diretamente um periférico, ocorre uma interrupção.** O sistema operacional é ativado, já em modo supervisor, e o processo de usuário é abortado, pois tentou um acesso ilegal. A única forma de o processo de usuário realizar uma chamada de I/O é através de uma chamada de sistema.

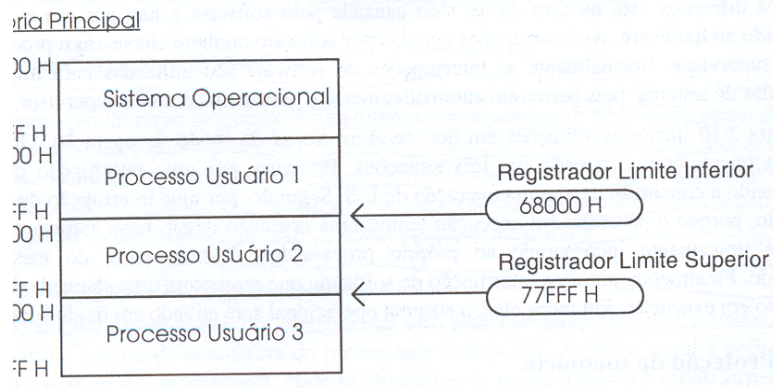
Muitas arquiteturas oferecem uma instrução chamada de interrupção de software ou trap. Essa instrução, quando executada, apresenta um efeito semelhante ao da interrupção por hardware. **As interrupções geradas por software também chaveiam o processador para modo supervisor.** Normalmente as interrupções de software são utilizadas para implementar as chamadas de sistema, pois permitem automaticamente a passagem pelo modo supervisor.

A figura mostra as situações onde ocorrem trocas do modo de operação. Observe que o sistema operacional é ativado em três situações. Primeiro, por **interrupção do periférico**, informando a conclusão de alguma operação de I/O. Segundo, por uma **interrupção do hardware de proteção**, porque o processo em execução tentou uma operação ilegal. Esse hardware de proteção aparece tipicamente incorporado ao próprio processador, fazendo parte do mesmo circuito integrado. Finalmente, por uma **interrupção de software**, que representa uma chamada de sistema do processo em execução. Em todas elas, o sistema operacional será ativado em modo supervisor.

Proteção da memória

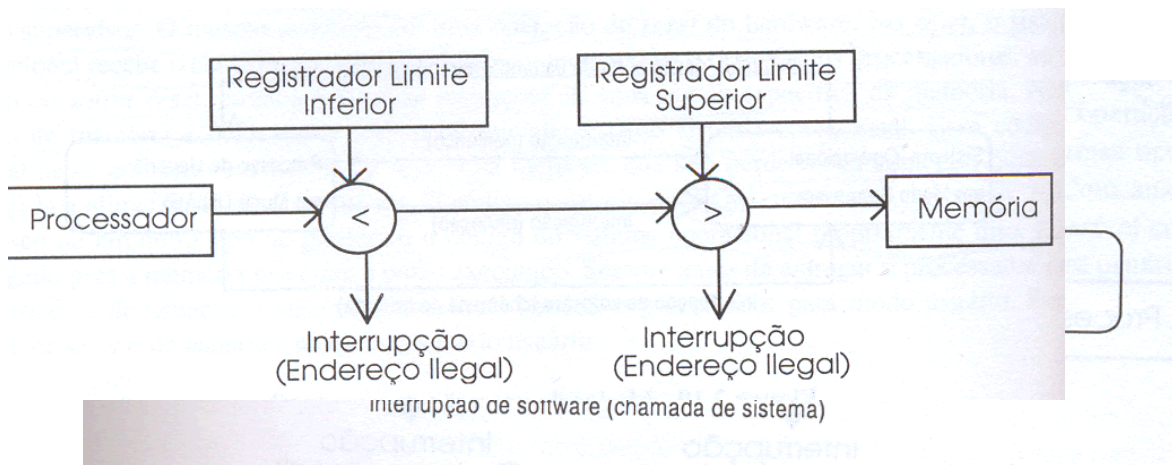
É necessário proteger a memória do sistema operacional, tanto código como dados. Ao mesmo tempo, **é necessário proteger a memória usada por um processo de acesso de outros processos.** Deve ser impossível para o processo do usuário João alterar a memória alocada para o processo da usuária Maria. Se isso ocorrer, os programas de Maria poderão não funcionar em decorrência de um comportamento indevido do processo de João.

Para implementar a proteção de memória, o sistema operacional também necessita de auxílio da arquitetura. **Sempre que o sistema operacional vai disparar um processo de usuário, entram em ação os registradores de limite, onde são carregados os valores relativos ao processo que vai**



executar. Ele coloca no registrador limite inferior o endereço do primeiro byte pertencente à área de trabalho do processo. No registrador limite superior, é colocado o endereço do último byte válido para a sua área.

A cada acesso à memória, o hardware de proteção compara o endereço gerado pelo processador com o conteúdo dos dois registradores de limite. Se o endereço gerado estiver **fora da área do usuário**, é gerada uma interrupção. Nesse caso, o sistema operacional é ativado em modo supervisor. **O processo de usuário será abortado, devido a um acesso ilegal à memória.**



Algumas arquiteturas fazem o acesso aos periféricos através de posições específicas da memória. Essa técnica é chamada de **I/O mapeada na memória**. Nesse caso, a proteção de memória também implementa a proteção dos periféricos.

Obviamente, o acesso aos registradores de limite deve ser feito através de instruções privilegiadas. Somente o sistema operacional pode alterar o seu conteúdo. Por outro lado, o sistema operacional deve acessar a área do usuário. Isso é necessário para o sistema operacional colocar lá o próprio programa do usuário. Muitas chamadas de sistema também fazem com que o sistema

operacional busque parâmetros ou coloque respostas diretamente na área do processo usuário.

Para evitar que um único processo de usuário monopolize a utilização do processador, é empregado um temporizador (timer). O temporizador é um relógio de tempo real, implementado pelo hardware, que gera interrupções de tempos em tempos. O período do temporizador corresponde ao intervalo de tempo entre interrupções. Em geral, ele é da ordem de milisegundos. As interrupções do timer ativam o sistema operacional. Ele então verifica se o processo executando já está muito tempo com o processador. Nesse caso, o sistema pode abortar o processo por exceder o tempo limite de execução. Essa é uma prática usual em sistemas dos quais a execução não é acompanhada pelo usuário, e um laço infinito no programa não seria percebido. Em outros sistemas é usual o processo voltar para a fila de aptos. Com isso, todos os processos teriam chance de serem executados e mais tarde o processo interrompido teria oportunidade novamente.

Todo o mecanismo de proteção está baseado em interrupções. O processo usuário deve sempre executar com as interrupções habilitadas. Somente assim uma operação ilegal será detectada.