Todas as questões relativas a P1 2004/1 estão com # na frente.

P1 2001

 # 1 - Existem 3 estratégias típicas para alocação dinâmica de processos em máquinas sem paginação. Digamos que estas estratégias se chamem A,B e C. Diga o nome verdadeiro para estas estratégias. Exemplifique, justificando, uma seqüência de alocação de forma que a estratégia A seja melhor que as outras. Exemplifique outra seqüência de forma que a estratégia B seja melhor. Finalmente exemplifique uma terceira seqüência onde a estratégia C seja a melhor.
Resposta:
As três estratégias são:
* Best-fit: procura alocar a área livre de tamanho mais próximo ao que se quer inserir, ou seja, varre a memória atrás da
área livre com o tamanho mais próximo do desejado.
* worst-fit: aloca a maior área livre de memória, ou seja, varre a memória atrás da maior área livre.
A intenção desse é a sobra de área livre contínua maior, uma vez alocado o desejado.
* First-fit: aloca a primeira área livre de memória que encontrar igual ou maior que a área a ser alocada. Costuma ser mais rápido, pois não precisa varrer toda a memória.

O problema da alocação é a fragmentação da memória, essas estratégias foram propostas com a intenção de evitar a fragmentação. Uma vez a memória fragmentada, a solução usada para desfragmentar é usar a compactação de memória.
Compactação de memória é agrupar as áreas livres de um lado e as áreas alocadas do outro lado, com isso gerando uma única área livre com todas as áreas livres anteriormente existentes.

O best-fit quando encontra uma área livre igual a procurada não gera fragmentação, caso contrário a área livre que sobra da área livre escolhida, pode ser pequena demais para alocar outro bloco. Já o worst-fit, alocando a maior área possível, essa área que sobra já pode ser maior, podendo armazenar outro bloco.
O first-fit, por alocar a primeira área de tamanho suficiente, pode gerar fragmentos de áreas livres de tamanho variável. Costuma ser mais rápido por não fazer a procura da mais adequada e sim pega logo a primeira área livre de tamanho igual ou superior.

Não existe um algoritmo pratico para a alocação.
O best-fit pode ser melhor que os outros quando existir áreas livres de tamanho igual ao das áreas que se deseja alocar, estando elas fora de ordem, ou seja, obtrigando o algoritmo procurar a área livre e não ser logo a primeira.
O worst-fit pode ser melhor que os outros quando as áreas maiores não estão em ordem, ou seja, obrigando o algoritmo procurar a maior, e a sobra das áreas livres alocadas, sejam grande o suficiente para serem alocadas por outros blocos.
O first-fit pode ser melhor que os outros algoritmos enquanto houver áreas livres para ser alocadas, em ordem, e a sobra das áreas livres alocadas, sejam grande o suficiente para serem alocadas por outros blocos.

 # 2 - Mostre o(s) mecanismo(s) de hardware envolvido(s) em:
2.1 - Tradução de endereço virtual para real
Resposta:
A tradução do endereço virtual para o real é feito pelo hardware através da TLB, quando não está na TLB, ele faz referência a memória para buscar o endereço virtual e o real correspondente.

2.2 - Falta de página
Resposta:
Quem gera falta de página é o hardware. Ao tentar converter a página virtual em página real, se o bit VÁLIDO da página não estiver setado, o hardware gera uma interrupção para o SO tratar e carregar a página se for o caso.

2.3 - Simulação de página LRU
Resposta:
Para facilitar a simulação LRU, a tabela de páginas tem um campo a mais que é o acessado. Toda vez que a página é acessada, o hardware vai na tabela e seta o bit. Um algoritmo que use a idéia da simulação LRU usa esse bit para saber
se a página foi acessada recentemente, se não estiver setado, essa página será escolhida como vítima.

2.4. - Copy on write de página
Resposta:

No copy on write o bit de readOnly está setado (a página lógica dos processos estão apontando para a mesma página física). Quando um processo tenta modificar essa página, o hardware verifica esse bit setado, não deixa o processo mudar e gera uma interrupção para o SO tratar. O SO por sua vez identifica que é um caso de copy on write e faz o tratamento adequado. O SO duplica a página, não seta o bit de readOnly e aponta a página lógica para essa nova área de memória.

2.5. - Carregamento de página sob demanda
Resposta:
No carregamento de página sobre demanda, o processo não vai todo para a memória, vai apenas a rotina principal. No decorrer da execução, as outras páginas podem ser necessárias, então, ao tentar converter a página virtual para real, o hardware verifica que a página está inválida e gera uma interrupção para o SO. O SO por sua vez trata essa interrupção, vê que a página pertence ao processo mas ainda não foi carregada. O SO carrega a página, troca o bit para válido e retoma a
execução ou coloca na fila de processos prontos.

3 - Mostre em "pseudo linguagem de máquina" como o código executado nas três formas típicas para transferência de dados entre os dispositivos e a memória. justifique.
Resposta:

 # 4 - Seja a seguinte seqüência de uso de páginas : 4 3 2 1 3 4 5 6 3 4 3 2 7 6 2 3 4 3 2 1. Se a memória real têm quatro páginas, quais são as faltas de página que ocorrem se forem utilizados os seguintes algoritmos de substituição de página (justifique): LRU, FIFO, Ótimo. Suponha a memória originalmente vazia.
Resposta:
* FIFO - a página vítima é aquela que foi carregada na memória há mais tempo.
* Ótimo - a página vítima é aquela que será usada em um futuro mais distante .
* LRU - a página vítima é aquela que foi menos recentemente usada.

P1 2002

 # 1 - O que é página vítima? Descreva um algoritmo ideal para a escolha da página vítima. E descreva um algoritmo utilizado na prática para essa escolha.
Resposta:
A página vítima é uma página escolhida pelo Software através da utilização de um algorítimo de substituição quando um processo requisita uma página real e não há mais nenhuma disponível.
O algoritmo ideal é chamado de ótimo e consiste em escolher como vítima a página que será utilizada em um futuro mais distante. Como não podemos adivinhar o futuro, não tem como implementar o ótimo. O que ele pode fazer é uma aproximação do ótimo, ao invés de pegar o mais distante no futuro, ele pega o que foi usado a mais tempo no
passado, o passado tem como o SO tomar conhecimento.

Um algoritmo utilizado é o LRU que escolhe como vítima a página que foi menos recentemente usada. Podemos implementar isso, usando a idéia do LRU, através do algoritmo do relógio para saber qual página foi menos recentemente usada. Este algoritmo é implementado com dois ponteiros que vão percorrendo a memória física com uma distância estabelecida (um atrás do outro) de forma circular. O ponteiro que vai na frente, vai verificando se o bit de acessado está
ligado, se estiver desligado, é essa página que ele pega como vítima. Já o segundo ponteiro, o que vem atrás, vai desmarcando o bite de acessado. Tem outros fatores que também pode ajudar na escolha da página vítima, tais como páginas que não foram alteradas e não precisam ser enviadas para disco, páginas de código por exemplo.

 # 2 - Quais são os bits de controle existentes na tabela de página? Descreva o motivo de cada um estar ligado ou desligado (se existir mais de um motivo, isso deve ser descrito).
Resposta:
Na tabela de páginas existe os seguintes bits de controle:
Válido, Executável, Alterável ou Read-Only, Núcleo e Acessado.
**** Válido***
Ligado: página pertence ao processo.
Desligado: a página pode não pertencer ao processo, a página pode ter sido escolhida como vítima, a página de código pode não ter sido carregada, a página de dado ou pilha pode nunca ter sido utilizada e pode ser usado para o software simular o bit de acessada quando o hardware não tem esse bit.
* Executável
Ligado / Desligado: a página contém ou não código executável.
* Alterável

Ligado: o conteúdo da página é alterável (área de dados ou de pilhas)
Desligado: o conteúdo da página não é alterável (área de código), no mecanismo de copy on write, o bit ALTERÁVEL OU READ-ONLY indica que a página está sendo compartilhada e não pode ser alterada no momento.
* Núcleo
Ligado / Desligado: página pertence ou não ao Sistema Operacional.
* Acessado
Ligado / Desligado: Bit utilizado na implementação do algoritmo de substituição de páginas da 2ª chance ou do relógio. Indica
se a página foi ou não acessada recentemente.

 # 3 - Descreva 3 estratégias para diminuir o problema de tabelas de páginas muito grandes.
Resposta:
* Geração de uma interrupção:
Nesta solução, quando o hardware tenta fazer a conversão (via TLB) e não consegue, ele gera uma interrupção para o software tratar e fazer a conversão, ou seja nesta solução quem faz a conversão, caso não esteja na TLB, é o software.
A conversão de virtual para real é complicada, devido a isso é feita pelo hardware. A idéia dessa solução é deixar livre a escolha da estrutura de controle, quem decide é o programador do SO.
* Tabela de página invertida:
A tabela de página invertida tem uma entrada para cada página real da memória. Cada entrada possui o endereço virtual da página armazenada naquela posição da memória real, com informações do processo proprietário da página. Assim, só existe uma tabela de página no sistema. Embora esse esquema reduza a quantidade de memória necessária para armazenar cada tabela de página, ele aumenta o tempo necessário para pesquisar a tabela quando ocorre uma referência de página, pois como a tabela é classificada por endereços reais e as pesquisas são feitas com endereços virtuais, às vezes a tabela precisa ser totalmente varrida para encontrar a tal referência.
* Uso de tabela em níveis:
Das entradas da tabela, a maior parte das linhas tem o bit de validade igual a 0, poucas tem o bit igual a 1. Logo, observou-se que forma uma estrutura de dados esparsa, que é uma estrutura de dados que contem campos com informações irrelevantes. A estrutura de dados esparsa, ao invés de fazer a matriz como um array direcional, faz um array de ponteiros para o array, nas poucas linhas que tem espaço relevante, essas linhas tem um array de linhas alocados. As demais linhas que tem sempre 0 não precisam de um array para guardar. Seguindo essa idéia, a tabela de páginas é dividida em partes e só irá existir as que tiver páginas validas. Terá uma tabela anterior que é chamada de diretório de páginas. Vão Ter pedaços iguais e só irá existir as tabelas que tem informações de páginas válidas, as demais páginas da tabela, apontam para o nil. Um problema é o aumento do tempo da conversão de virtual para real, são gastos 2 acessos a tabela de páginas, 1 para acesso ao diretório e outro a operação da tabela de páginas. Quando a conversão não está na TLB, esse é bem maior, pois terá 3 acessos a Tabela de páginas. Apesar desse problema de ficar mais lento, isso resolve o problema de tabelas
grandes.

 # 4 - Quais são as áreas de memória típicas de um processo? O que é localizado em cada uma dessas áreas? O que motiva o crescimento de cada uma dessas áreas? Como este crescimento pode ocorrer com segmentação? Com paginação?
Resposta:
As áreas de memória típicas são:
* Área de pilha (reservada para variáveis locais)
* Área de dados (para as variáveis globais e dinâmicas)
* Área de código (código do programa)

Códigos e dados são alocados quando o processo começa a executar.
Dados locais e variáveis dinâmicas vão sendo alocados conforme o programa executa.
A Diferença entre variáveis locais e dinâmicas, é que As Variáveis locais em um seqüênciamento muito claro de alocação e desalocação quando a rotina começa a rodar, são alocadas todas as variáveis locais para essa rotina, quando a rotina acaba de rodar, são desalocados na mesma ordem em que foram alocadas. Nas variáveis dinâmicas o programador pode escolher a ordem de desalocação, não seguem a ordem de pilha como nas locais. Não dá para usar uma pilha para alocar variáveis dinâmicas. A grande diferença é o sequenciamento.
A área de código pode ser modificada durante a execução do programa (programas automodificáveis), esses programas geram códigos que são incorporados a eles mesmos, mas isso não é muito comum. Geralmente a área de código fica protegida, se o processo tentar modifica-la ele é abortado. Nos sistemas operacionais existentes hoje em dia não existe a possibilidade de alterar a área de código.

A área de pilha e a área de dados sofrem modificação, elas são alteráveis. Na área de pilha pode ocorrer o stack overflow,
que é o estouro da pilha, ou seja, quando o topo alcança o limite da pilha. Quando ocorre isso tem SO que tenta aumentar o limite da pilha (UNIX), caso contrário aborta o processo. A segmentação utiliza uma tabela de segmentos (guarda o início e o tamanho do segmento), dessa forma, visualiza a memória unidimensional de uma forma bidimensional. Com isso, a alocação não precisa ser continua. A segmentação possibilita a expansão da pilha, com segmento de pilha o hardware é capaz de perceber que o topo está no seu limite inferior. Ele sabe o tamanho do segmento, levanta interrupção e o SO trata e testa o tamanho da pilha. Logo, a segmentação possibilita o reconhecimento do estouro da pilha. Com a paginação passou a existir duas formas de ver a memória, o espaço de endereçamento lógico e o espaço de endereçamento físico. Cada processo tem seu espaço de endereçamento lógico, ou seja, ele pensa que está sozinho e a memória é toda dele. Com a paginação, basta ter memória real livre em qualquer posição que será mapeada na memória lógica. Esse espaço de endereçamento lógico é muito grande, praticamente acaba com o problema do crescimento da área de dados e de pilha, bastando para isso colocá-las bem afastadas uma da outra, logo enquanto houver memória física, não existirá problema em crescer a área de dados para cima e a área de pilha para baixo. Se alguma delas crescer e bater na outra, não tem jeito, o processo será abortado mesmo existindo memória física.

P1 2003/1

 # 1 - Diferencie overlay, swap de processos e paginação sob demanda. Quais são seus propósitos? Quais as vantagens de um sobre o outro?
Resposta:
* Overlay é um mecanismo utilizado para permitir que o tamanho de um processo seja maior do que a área de memória alocada a ele. A idéia é dividir o programa em partes que não necessitem estar carregadas na memória ao mesmo tempo, e carregar uma parte fixa que pode ser trocado por outra conforme necessário. É um mecanismo simples, que pode ser implementado em qualquer Software, mas que requer um grande esforço por parte do programador para ser implementado.
Swap de processos é um mecanismo que permite que a soma das áreas ocupadas pelos processos que estão sendo executados seja maior do que a memória física disponível. Consiste na transferência de um processo da memória para o disco, devido a necessidade de alocação de um novo processo e a falta de memória suficiente para isso. Onde o processo transferido pode retornar a memória mais tarde para continuar sua execução. O tempo gasto nas transferências de processos é relativamente grande, tem o problema de fragmentação de memória também.
* A paginação sob demanda permite que só estejam carregadas na memória as páginas do processo que estão realmente sendo utilizadas ou que foram utilizadas há pouco tempo. Consiste na divisão da memória e do processo em páginas do mesmo tamanho que são mapeadas através de uma tabela de páginas. Sempre que uma página do processo tiver de ser acessada e não estiver presente na memória, será gerada uma interrupção de hardware para verificar se a página pertence ao processo e então carregá-la na memória. Logo, necessita de suporte de hardware para funcionar. Caso não exista espaço disponível, o software deverá escolher uma página da memória física para ser colocada em disco (ou não) dando lugar a página solicitada. Uma das vantagens da paginação sob demanda é que a troca é feita entre páginas do processo e não entre processos inteiros, diminuindo assim o tempo de transferência. Praticamente não tem fragmentação, passa a ter somente fragmentação interna na página que é irrelevante. Uma grande vantagem é o fato de não ocupar memória com coisas que pode nunca ser usada.

 # 2- Exemplifique, explicando, quatro algoritmos diferentes de escolha de página vítima. Utilize a mesma sequência de páginas sendo acessadas.
Resposta:
Seqüência: 4, 1, 2, 3, 1, 4, 1, 6, 4, 1, 4, 3, 2, 3, 4, 1, 3
Memória com três páginas

* FIFO - a página vítima é aquela que foi carregada na memória há mais tempo.
_ _ _ F: entra 4
4 _ _ F: entra 1
4 1 _ F: entra 2
4 1 2 F: sai 4 e entra 3
3 1 2 : 1, F: sai 1 e entra 4
3 4 2 F: sai 2 e entra 1
3 4 1 F: sai 3 e entra 6
6 4 1 :4, 1, 4, F: sai 4 e entra 3
6 3 1 F: sai 1 e entra 2
6 3 2 :3, F: sai 6 e entra 4
4 3 2 F: sai 3 e entra 1

4 1 2 F: sai 2 e entra 3
4 1 3
Total de faltas: 12

* Ótimo - a página vítima é aquela que será usada em um futuro mais distante .

* LRU - a página vítima é aquela que foi menos recentemente usada.

* Algorítimo da Segunda chance - utiliza um bit de acessado, um ponteiro vai passando em cada página, se o bit não estiver setado essa será escolhida como vítima. Se o bit estiver setado, a página ganha uma segunda chance, não é escolhida como vítima, seu bit de acessado fica não setado e o ponteiro passa adiante procurando uma página com o bit de
acessado não setado.

 # 3 - Mostre a(s) tabela(s) de página(s) (com seus bits de conytrole) nas seguintes situações:
3.1 - Copy on write (antes e depois da alteração do conteúdo de uma página virtual.
Resposta:
Antes os processos estão compartilhando a página física que não pode ser alterada, logo o bit de readOnly está setado.
Quando um dos processos tenta alterar a página, não consegue, pois a página está readOnly. então, ele realmente faz a cópia da página, copia para uma outra área de memória livre, não seta o bit de readOnly. Só nessa hora que realmente é feita a cópia e o processo já pode alterar a página.

3.2 - Memória compartilhada entre processos.
Resposta:
Se os processos não poderem alterar a página, o bit de readOnly vai estar setado, caso contrário não vai estar setado.

 # 4 - O que é amarração de endereços (address binding)? Quais são os diferentes momentos em que a amarração pode ser realizada? Explique.
Resposta:
Address binding é associação feita entre os endereços lógicos de instrução e dados de um programa e endereços físicos da
memória. Pode ser realizada em três diferentes etapas da vida de um programa:
* Em tempo de compilação: cabe ao compilador definir o local da memória onde o programa será carregado. Se houver a necessidade de se mudar o local de armazenamento, o programa terá de ser recompilado.
* Em tempo de carga (endereço absoluto): o carregador deve acertar as referências de acordo com o endereço inicial onde o programa será carregado uma vez carregado não pode ser mudado de posição na memória.
* Em tempo de execução (uso de endereços relativos): a correção do endereço é feita durante a execução. É necessário hardware especial para suportar esse esquema. Tem dois casos:
- Relativo ao registrador de base
O sistema operacional não faz nada com relação à correção de endereço, mas a cada instrução executada a CPU soma o valor do endereço absoluto da instrução mais o endereço contido no registrador de base. O responsável pela alteração do endereço do registrador de base é o sistema operacional. A vantagem desta solução para a solução anterior é que o executável não precisa ter nenhuma informação de controle de endereço, simplificando a carga. Quanto ao tempo de execução em teoria seria mais lento devido a cada instrução que faça referência a endereços absolutos ter uma soma, porém esta soma é feita pelo hardware, um hardware dedicado a fazer isto. Com isso o tempo que leva esta soma é irrelevante, não contribuí para que CPUs que usem registrador de base sejam mais lentas que CPUs que não usem registrador de base. Se fosse uma operação feita por software poderia ficar mais lento, mas não é o caso.
A CPU Intel tinha um problema que a distância em relação à base não podia ser muito grande, a limitação era 64Kbytes, à
distância do endereço da base. Se o programa ".EXE" tivesse menos que 64Kbytes, então, podia usar apenas esta solução. Caso o programa tivesse mais que 64Kbytes, então, tinha que usar uma solução mista.
- Relativo a instrução corrente
São endereços que contam a partir da própria instrução. Nesta solução não importa onde o programa foi carregado na memória, pois a distância de uma instrução desvio para o seu destino, não se altera. O código que só usa este tipo de endereço é chamado código relocável, porque ele pode ser mudado de posição na memória e continuar executando sem precisar fazer mais nada.

 # 5 - Descreva a alocação de páginas global e por processo(local). Descreva os problemas que existem em cada uma das abordagens.
Resposta:

Na Alocação global, não existe número de páginas por processo, qualquer página de qualquer processo pode ser escolhida como vítima, o processo pode receber páginas físicas de outros processos que foram escolhidas como vítima. Já na alocação local, existe um número de páginas por processo, ou seja, cada processo só pode receber como página vítima, páginas entre aquelas que foram destinadas a ele. Falta de páginas de um processo que necessite de página vítima, a página vítima tem que ser entre as destinadas a esse processo. O Windows NT tem um número mínimo e um número máximo de páginas, o processo não pode ter menos que o mínimo e mais que o máximo. A dificuldade do algoritmo é para saber o número de páginas físicas que devem ser alocada para cada processo. Este número tem que ser maior ou igual ao workingset do processo, número de páginas físicas que o processo está usando em um certo instante de tempo. Se o SO conseguir alocar um número de páginas maior ou igual ao workingset do processo, a falta de página quase não vai existir, vai existir falta de página quando trocar o workingset do processo. Se o número de páginas físicas for menor que o workingset do processo, vai ocorrer falta de página o tempo todo, pois uma página que foi escolhida como vítima deverá ser usada logo em seguida, como não está na memória vai gerar falta de página. Não tem como o SO saber o workingset do processo, o que ele faz é observar o número de falta de página do processo, se esse número for grande, ele fica sabendo que o workingset do processo é maior que o número de páginas físicas que o processo tem. Para resolver isso, ele aumenta o número de páginas físicas desse processo. Com isso, enquanto esse workingset não mudar e se já estiver todo na memória, não vai ocorrer falta de página para esse processo. A falta de página só ocorre quando trocar o workingset. Por outro lado, se o SO observar que o processo tem mais páginas físicas que o workinkset, ele pode diminuir o número de páginas físicas alocadas ao processo. O SO acompanha o número de falta de página dos processos. Uma coisa que pode acontecer, é a máquina ficar muito lenta motivada por excesso de falta de página, 1% de falta de página já é um número muito grande e o suficiente para acontecer isso. Isso pode acontecer quando há processo que aloca e usa muita memória.
Na alocação global, as páginas dele acaba roubando páginas de outros processos, já que não tem número de páginas de memória física reservada a cada processo. Como conseqüência, vai acontecer dos outros processo gerar falta de página também. Esta situação da máquina ficar lenta devido a falta de página é chamada de thrash. Uma causa para isso pode ser um processo que necessita de muita memória. Outra coisa que pode causar thrash é muitos processos em execução, tendo muitos processos que alocam e usam, acaba acontecendo de ocupar toda a memória da máquina. Processos típicos não usam a CPU o tempo todo (trocam de estado entre executando e bloqueado), quanto mais processos em execução, mais vai ocupar a CPU, até chegar a 100% do uso da CPU com o aumento do número de processos. O descrito acima não se aplica a processos que ocupam a CPU o tempo todo, neste caso basta um processo para usar a CPU 100%. Isso ocorre se não considerar a existência de falta de página. Com a ocorrência de falta de página, existe um número máximo de processos em execução, passando desse número, como a memória não é grande o suficiente, os processos vão ficar mais tempo paginando e conseqüentemente o uso da CPU cai. Com a falta de página a CPU nunca é usada 100%.
Nesse ponto que o uso da CPU cai é onde fica caracterizado o Thrash, onde a máquina começa a ficar lenta. Nesse ponto, quando coloca mais processo, mais a máquina fica lenta, menos CPU é usada. O thrash pode acontecer nas duas formas de alocação, mas é mais comum na alocação global de páginas físicas, pois nesse tipo de alocação, pode acontecer de ter um único processo que use muita memória. Se ele usar toda memória física da máquina, os outros processos não vão ter memória e vão ocasionar o thash. Com alocação local, isso não acontece tão facilmente, pois cada processo tem o seu número de páginas físicas reservado. Cada processo normalmente tem o número mínimo e máximo de páginas reservadas, se o processo alcançar o máximo, ele fica lento, mas os outros não. Com o tempo acaba provocando thrash, pois o SO vai dar mais páginas para esse processo, mas é uma coisa mais controlada.
O thrash ocorre mais facilmente em alocação global. O UNIX resolve o problema do thrash com swap de processo, ou seja, ele tira completamente o processo da memória, com isso, liberando páginas físicas para os demais processos baixando o número de processos da memória. Quando se tem n processos, mas nenhum pronto para executar, o uso da CPU não é total. Basta um processo estar em execução para o uso da CPU ser total. Para o uso da CPU ser total o ideal é que tenha sempre processos prontos para execução. A falta de página provoca thrash porquê bloqueia o processo, poucas faltas não afetam muito, mas se as faltas forem muitas podem provocar pouco uso da CPU.

P1 2003/2

 # 1- Qual o problema é resolvido (e como é resolvido) pelo overlay? É necessário alguma facilidade especial do hardware para a implementação do overlay? Qual? Cite, explicando, uma outra solução para o mesmo problema que NÃO necessite de facilidade de hardware.
Resposta :
O problema resolvido é a falta de memória física para alocar um processo, ou seja, a quantidade de memória que o processo precisa é maior que a quantidade de memória disponível. Com o uso de overlay, a quantidade de memória que um processo precisa pode ser maior que a quantidade de memória física disponível. O método consiste em dividir um programa em partes de tamanho fixo (overlays) e carregar na memória algumas dessas

partes juntamente com uma rotina de gerenciamento de troca. Quando há a necessidade de carregar um novo overlay, a rotina de gerenciamento a carrega no lugar de um outro overlay que não seja mais necessário.
Overlay é um mecanismo muito simples e que não precisa de suporte especial do hardware, pois todo o trabalho de gerenciamento de troca é feito por software.
O swap de processo é uma outra solução que não necessita de suporte especial do hardware. Consiste em transferir da memória para o disco um processo que não esteja em uso , dessa forma liberando espaço de memória para um outro processo que esteja necessitando. Dessa mesma forma também, o processo que foi levado para o disco pode voltar para a memória caso necessite de continuar sua execução. A transferência é gerenciada pelo sistema operacional, pode ser
provocada devido a necessidade de liberar memória para um processo de maior prioridade.

 # 2- Qual a relação entre falta de página e página vítima? Essa relação sempre existe? Por quê?
Resposta :
Falta de página ocorre quando o hardware encontra o bit de inválido ligado na tabela de páginas, que significa para o hardware que a página não está carregada em memória naquele momento. Página vítima é o nome que se dá a página escolida para sair da memória, liberando espaço para outra página . Essa relação existe, pois se a memória estiver toda ocupada e ocorrer uma falta de página, o SO tem que escolher uma página vítima para dar lugar a essa página que gerou a falta. Na realidade, o SO sempre deixa algumas páginas livres, quando pode escolhe páginas vítimas. Dessa forma otimiza o tempo, ou seja, o tempo que levaria para escolher uma página vítima e fazer todo processo necessário de tratamento dessa página, não seria junto com o carregamento da página que gerou a falta em cituações que a memória estária toda ocupada. Com isso nem sempre a falta de página vai startar a escolha de página vítima. Se há páginas livres, necessariamente não precisa escolher página vítima para gravar em disco.

3- Porque o poling tem um loop de espera? Qual é (e como é) o mecanismo de hardware que possibilita existi uma melhor forma de resolver o problema.
Resposta:

4- A) Diga quais são, explicando, os três tipos de tempo gastos na leitura de um disco. **B)** Diga quais são, explicando, as três dimensões que precisam ser identificadas
para o acesso ao disco. Explique como este acesso tridimensional pode ser simplificado (use o conceito de cilindro).
Resposta da letra A : O tempo gasto na leitura de um disco é a soma de três tempos:
Tempo de seek: É o tempo gasto para que a cabeça de leitura se desloque até a trilha onde está o setor a ser lido.
Tempo de rotação: É o tempo gasto para que o setor a ser lido passe sob a cabeça de leitura.
Tempo de latência: É o tempo gasto na transferência de um conjunto de bytes do disco para a memória.

 # 5 - Processos podem se comunicar através de troca de mensagens. O envio de uma mensagem de um processo para outro processo costuma ser implementado através da cópia do conteúdo da mensagem que está em um processo para uma variável do outro processo. Como seria possível, com a paginação, fazer o envio da mensagem sem a cópia do conteúdo da mensagem.
Resposta:
Através do compartilhamento de páginas a mensagem pode ser passada, fazendo o processo remetente desmapear a(s) página(s) que contém a mensagem e o processo receptor mapear. Dessa forma somente os nomes de página têm de ser copiados, em vez de todos os dados. Outra técnica é a memória compartilhada distribuída. Pode permitir que múltiplos processos sobre uma rede compartilhem um conjunto de páginas, como um único espaço de endereço linear compartilhado. Quando um processo referencia uma página que atualmente não está mapeada, ele obtém uma falha de página. O manipulador de falha de página, que pode estar no kernel ou no espaço de usuário, então, localiza a máquina que armazena a página e envia uma mensagem solicitando que ele desmapeie a página e envie pela rede. Quando a página chega, ela está mapeada, e a instrução que está falhando é reiniciada.

PF 2003/1

1. Mostre com linhas de código como pode ser feita a impressão na tela dos caracteres 1025 a 2048 de um arquivo em duas situações:
a. chamadas convencionais de manipulação de arquivo.
b. arquivo mapeado em memória.

Faça a impressão no formato: "O caracter %d do arquivo é o %c \n"
Qual a vantagem de uma forma sobre a outra em termos de velocidade?

Por que?

2. Seja três arquivos, um com 13 blocos, outro com 12 e outro com 6. Cada um desses arquivos tem 3 pedaços contínuos. Para os mesmos arquivos mostre como fica a estrutura de alocação nos seguintes casos:
a. Alocaçõa com FAT.
b. Alocação indexada do unix.
c. Alocação por extensão do NTFS.

Para cada caso, mostre também a estrutura de gerência de blocos livres quando necessário.

 # 3. Sejam 3 processos criados a partir do mesmo programa, cada áreas dos processos tem pelo menos 3 páginas. 2 desses processos foram iniciados por usuários diferentes, o terceiro foi criado por FORK() (criação de processos no UNIX) a partir de um dos dois primeiros. Suponha que todas as páginas estejam presente na memória física. Mostre como ficam as
tabelas de páginas com os bits de válido e read-only para esses três processos.
Resposta:
Os 2 processos iniciados por usuários diferentes, como são do mesmo programa, a princípio terão os bits de válido setado e o de read only também setado, pois estarão compartilhando as páginas. Quando for criado o terceiro processo, ele vai a princípio compartilhar as páginas de memória com o processo que o criou. Com isso o bit de valido do processo criador e do criado, ficará setado e o bit de read only desses dois processos ficarão setados também. Com o bit de read onlh setado, quando um dos processos que compartilham memória tentar alterar a página, o hardware vai gerar uma interrupção para o sistema operacional. Em seguida o sistema operacional vai tratar essa interrupção. Ele vê que a página pertence ao processo, logo ele não bloqueia o processo, ele duplica aquela área de memória, aponta o processo que tentou modificar aquela página para a nova área, não seta o bit de read only e desbloqueia o processo colocando na fila de prontos para ser escalonado.

4. Qual a relação existente entre escalonamento de disco e de processos. Quais são dois critérios que norteiam os algorítimos de escalonamento de disco?
Mostre, exemplificando, como estes critérios estão presentes nos algoritmos.

PF 2003/2

1) Seja um disco com 200 cilindros, para este disco chegam pedidos de leitura para os cilindros 55, 58, 39, 18, 90, 160, 150, 38 e 184 (nesta ordem). Dado que a cabeça está atualmente no cilindro 100 indo para o 200, mostre como os pedidos serão atendidos utilizando os seguintes algoritmos:
a) FCFS
Resposta:
Esse algoritmo atende a fila de pedido na forma que ela está (pode ocorrer caso desse escalonamento atender pedidos que movimentem muito a cabeça do disco, e não pedidos próximos).
Atendimento: 55, 58, 39, 18, 90, 160, 150, 38 e 184

b) SSTF
Resposta:
Este algoritmo seleciona o pedido com o tempo de busca mínimo a partir da posição atual da cabeça. Como o tempo de busca aumenta com o número de cilindros percorridos pela cabeça, o SSTF escolhe o pedido pendente mais próximo da posição atual da cabeça. O problema que pode acontecer é a ocorrência de Estarvation (paralização), como a chegada de pedido de acesso a disco é uma coisa que pode acontecer todo tempo, se os pedidos que estão entrando na fila forem pedidos próximo a posição atual da cabeça, ele pode ficar atendendo a esses pedidos, fazendo que um pedido velho mais longe da cabeça fique esperando eternamente.
Atendimento: 90, 58, 55, 39, 38, 18, 150, 160 e 184

c) Elevador
No algoritmo do elevador, o braço do disco começa em uma ponta do disco e se movimenta em direção à outra ponta, atendendo os pedidos assim que chega em cada cilindro, até atingir a outra ponta do disco. Na outra ponta, o sentido do movimento da cabeça é invertido percorrendo o disco no outro sentido. Ao chegar na ponta e voltar, no início dessa volta tem poucos pedidos, pois a cabeça do disco acabou de passar por ali, ficando um maior número de pedidos na outra ponta.
Atendimento: 150, 160, 184, 90, 58, 55, 39, 38 e 18

 # 2) Suponha que a tabela de páginas de uma máquina esteja contida toda em registradores. Suponha os seguintes tempos:
? 8 milissegundos para tratar uma falta de página quando existem páginas físicas livres;
? 20 milissegundos para tratar uma falta página quando é necessário fazer a substituição de página;
? 100 nanossegundos para acessar a memória física.
Suponha que em 70% das faltas de página, a página tenha que ser substituída. Mostre, explicando, qual a maior taxa para falta de páginas de forma que o tempo efetivo de acesso à memória seja de 200 nanossegundos.
Resposta:
O tempo efetivo de acesso a memória é igual ao tempo de acesso a memória física vezes a probabilidade da página está em
memória, mais o tempo de carregamento em memória vezes a probabilidade de não estar na memória.
P = taxa para falta de página
m = tempo na memória = 100000000
f = falta de página = (8 * 0.3) + (20 * 0.7) = 16.4
200000000 = ((1- p) * m)+ (p * f)
100000000 = -100000000p + 16.4p
...
 # 3) Responda se as afirmativas seguintes são verdadeiras ou falsas, justificando sua resposta:
? Para todos os processos, existe um número tal que, se um processo tenha menos páginas físicas do que este número, ele executará mais lentamente, caso ele tenha mais páginas físicas que este número sua velocidade de execução praticamente não sofrerá alteração.
Resposta:
Essa afirmativa é verdadeira.
Esse número é o número de páginas do workingset, que é o conjunto de páginas que o processo em um certo instante de tempo precisa para não ter muita falta de página. Se o Sistema operacional oferecer a ele um número de páginas físicas menor que o número de páginas do workingset, ele vai gerar um número grande de falta de página, com isso vai rodar mais lento. Através desse número de falta de página, o Sistema Operacional toma conhecimento da quantidade de páginas física que o processo precisa. Se o número for pequeno quer dizer que o processo tem mais páginas físicas que o número
de páginas do workingset, ou seja, mais páginas do que preciza. Logo, se tiver mais páginas que o número de páginas do workingset, a velocidade quase não sofre alteração, pois vão ficar páginas físicas sem uso.

? O tamanho do Working Set de um certo processo é alterado dependendo do número de processos que estão em execução.
Resposta:
Esta afirmativa está errada, pois o workingset do processo é o conjunto de páginas fisicas para cada processo em um certo instante de tempo. O que muda é o número de páginas físicas oferecidas para cada processo. Quando entra um processo novo e não tem páginas físicas para ele, o Sistema Operacional pega um número igual de páginas de cada processo e
dá para esse processo novo. Se com isso acontecer de ocorrer muitas faltas de páginas, o Sistema Operacional escolhe um processo que não esteja em execução para fazer shuap e redistribui as páginas dele para os outros processos.

4) Considere a alocação de arquivos do Unix. Assuma que os blocos têm 4KB e cada ponteiro para bloco tenha 8 bytes. Diga, justificando, qual é o maior tamanho possível de arquivo que pode ser gerenciado (deixe os cálculos indicados).
Resposta
Utilizando blocos de 1Kb cada bloco de indireção terá 256 entradas. Logo com blocos de 4Kb cada bloco de indireção terá 1024 entradas. Somando as 10 entradas contidas no inode, usando somente bloco de indireção um arquivo pode ter no máximo 1034 blocos. Utilizando bloco de dupla indireção o arquivo pode ter no máximo:
10+ 1024 + 1024*1024 blocos.
Utilizando bloco de tripla indireção o arquivo pode ter no máximo:
10 + 1024 + 1024*1024 + 1024*1024*1024 blocos
O maior tamanho de arquivo suportado é a conta usando o bloco de tripla indireção vezes 1000.

5) Para cada uma das operações abaixo, diga, justificando, se o inode necessita ser alterado (considere o arquivo não esparso e desconsidere a data e hora de última alteração):
? Leitura de um bloco no meio do arquivo;
Resposta:
Não preciza ser alterado pois não cresceu e nem diminuiu o tamanho do arquivo.

? Escrita em um bloco no meio do arquivo;

Resposta:
O inode pode crescer dependendo da quantidade de informação da escrita.

? Escrita após o final de um arquivo de uma informação cujo tamanho é maior que um bloco;
Resposta:
Dependendo da quantidade de crescimento pode crescer usando bloco de indireção, dupla indireção ou tripla indireção

? Escrita após o final de um arquivo de uma informação com 1 byte de tamanho.
Resposta:
Cresce usando bloco de indireção.

PR 2003/2

 # 1) Como processadores de 64 bits resolvem o problema de mapeamento do espaço de endereçamento lógico para o espaço de endereçamento físico. Mostre duas soluções diferentes.
Resposta:
- Uma solução é o uso de tabela em níveis.
Das entradas da tabela, a maior parte das linhas tem o bit de validade igual a 0, poucas tem o bit igual a 1. Logo, observou-se que forma uma estrutura de dados esparsa, que é uma estrutura de dados que contem campos com informações inrelevantes. A estrutura de dados esparsa, ao invés de fazer a matriz como um array direcional, faz um array de ponteiros para o array, nas poucas linhas que tem espaço relevante, essas linhas tem um array de linhas alocados. As demais linhas que tem sempre 0 não precisam de um array para guardar. Seguindo essa idéia, a tabela de páginas é dividida em partes e só irá existir as que tiver páginas validas. Terá uma tabela anterior que é chamada de diretório de páginas. Vão Ter pedaços iguais e só irá existir as tabelas que tem informações de páginas válidas, as demais páginas da tabela, apontam para o nil. Um problema é o aumento do tempo da conversão de virtual para real, são gastos 2 acessos a tabela de páginas, 1 para acesso ao diretório e outro a operação da tabela de páginas. Quando a conversão não está na TLB, esse é bem maior, pois terá 3 acessos a Tabela de páginas. Apesar desse problema de ficar mais lento, isso resolve o problema de tabelas grandes.

- Uma outra solução seria o uso de tabela de página invertida.
A tabela de página invertida tem uma entrada para cada página real da memória. Cada entrada possui o endereço virtual da página armazenada naquela posição da memória real, com informações do processo proprietário da página. Assim, só existe uma tabela de página no sistema. Embora esse esquema reduza a quantidade de memória necessária para armazenar cada tabela de página, ele aumenta o tempo necessário para pesquisar a tabela quando ocorre uma referência de página, pois como a tabela é classificada por endereços reais e as pesquisas são feitas com endereços virtuais, às vezes a tabela precisa ser totalmente varrida para encontrar a tal referência.

2) Para um dispositivo como um mouse que gera poucos bytes de informação em uma periodicidade aleatória, qual a melhor forma de ser feita a entrada/saída ? Justifique. Por que as duas outras formas não são interessantes neste caso ?
Resposta
A melhor forma é usando interrupção de hardware.

 # 3) Descreva os dois tipos de alocação de páginas físicas. Como o thrashing pode ser resolvido em cada um deles.
Resposta:
A alocação de páginas físicas pode ser global ou local.
Na Alocação global, não existe número de páginas por processo, qualquer página de qualquer processo pode ser escolhida como vítima, o processo pode receber páginas físicas de outros processos que foram escolhidas como vítima. Já na alocação local, existe um número de páginas por processo, ou seja, cada processo só pode receber como página vítima, páginas entre aquelas que foram destinadas a ele. Falta de páginas de um processo que necessite de página vítima, a página vítima tem que ser entre as destinadas a esse processo. O Windows NT tem um número mínimo e um número máximo de páginas, o processo não pode ter menos que o mínimo e mais que o máximo. A dificuldade do algoritmo é para saber o número de páginas físicas que devem ser alocada para cada processo. Este número tem que ser maior ou igual ao workingset do processo, número de páginas físicas que o processo está usando em um certo instante de tempo. Se o SO conseguir alocar um número de páginas maior ou igual ao workingset do processo, a falta de página quase não vai existir, vai existir falta de página quando trocar o workingset do processo. Se o número de páginas físicas for menor que o workingset do processo, vai ocorrer falta de página o tempo todo, pois uma página que foi escolhida como

vítima deverá ser usada logo em seguida, como não está na memória vai gerar falta de página. Não tem como o SO saber o workingset do processo, o que ele faz é observar o número de falta de página do processo, se esse número for grande, ele fica sabendo que o workingset do processo é maior que o número de páginas físicas que o processo tem. Para resolver isso, ele aumenta o número de páginas físicas desse processo. Com isso, enquanto esse workingset não mudar e se já estiver todo na memória, não vai ocorrer falta de página para esse processo. A falta de página só ocorre quando trocar o workingset. Por outro lado, se o SO observar que o processo tem mais páginas físicas que o workinkset, ele pode diminuir o número de páginas físicas alocadas ao processo. O SO acompanha o número de falta de página dos processos. Uma coisa que pode acontecer, é a máquina ficar muito lenta motivada por excesso de falta de página, 1% de falta de página já é um número muito grande e o suficiente para acontecer isso. Isso pode acontecer quando há processo que aloca e usa muita memória.
Na alocação global, as páginas dele acaba roubando páginas de outros processos, já que não tem número de páginas de memória física reservada a cada processo. Como conseqüência, vai acontecer dos outros processo gerar falta de página também. Esta situação da máquina ficar lenta devido a falta de página é chamada de thrash. Uma causa para isso pode ser um processo que necessita de muita memória. Outra coisa que pode causar thrash é muitos processos em execução, tendo muitos processos que alocam e usam, acaba acontecendo de ocupar toda a memória da máquina. Processos típicos não usam a CPU o tempo todo (trocam de estado entre executando e bloqueado), quanto mais processos em execução, mais vai ocupar a CPU, até chegar a 100% do uso da CPU com o aumento do número de processos. O descrito acima não se aplica a processos que ocupam a CPU o tempo todo, neste caso basta um processo para usar a CPU 100%. Isso ocorre se não considerar a existencia de falta de página. Com a ocorrência de falta de página, existe um número máximo de processos em execução, passando desse número, como a memória não é grande o suficiente, os processos vão ficar mais tempo paginando e conseqüentemente o uso da CPU cai. Com a falta de página a CPU nunca é usada 100%. Nesse ponto que o uso da CPU cai é onde fica caracterizado o Thrash, onde a máquina começa a ficar lenta. Nesse ponto, quando coloca mais processo, mais a máquina fica lenta, menos CPU é usada. O thrash pode acontecer nas duas formas de alocação, mas é mais comum na alocação global de páginas físicas, pois nesse tipo de alocação, pode acontecer de ter um único processo que use muita memória. Se ele usar toda memória física da máquina, os outros processos não vão ter memória e vão ocasionar o thash. Com alocação local, isso não acontece tão facilmente, pois cada processo tem o seu número de páginas físicas reservado. Cada processo normalmente tem o número mínimo e máximo de páginas reservadas, se o processo alcançar o máximo, ele fica lento, mas os outros não. Com o tempo acaba provocando thrash, pois o SO vai dar mais páginas para esse processo, mas é uma coisa mais controlada. O thrash ocorre mais facilmente em alocação global. O UNIX resolve o problema do thrash com swap de processo, ou seja, ele tira completamente o processo da memória, com isso, liberando páginas físicas para os demais processos baixando o número de processos da memória.
Quando se tem n processos, mas nenhum pronto para executar, o uso da CPU não é total. Basta um processo estar em execução para o uso da CPU ser total. Para o uso da CPU ser total o ideal é que tenha sempre processos prontos para execução. A falta de página provoca thrash porquê bloqueia o processo, poucas faltas não afetam muito, mas se as faltas forem muitas podem provocar pouco uso da CPU.

4) A) Dado um arquivo de 20KB armazenado de forma descontínua no disco, mostre uma seqüência de chamadas que o torne esparso (diga o tamanho do bloco considerado).
B) Mostre como ficaria a estrutura de alocação final do arquivo no caso de: i) Alocação por extensão ii) FAT
Resposta:
A)
B)
Por extensão:
Bloco inicial da extensão -- Tamanho da extensão -- Posição do bloco no arquivo lógico
10 -- 20 -- 1
32 -- 1 -- 21

PR 2002/1

 # 1 - Explique 4(quatro) formas que permitam carregar um programa em qualquer lugar da memória.
Resposta:
Essas quatro formas são as soluções para resolver o problema de amarração de endereço:
* Amarração em tempo de carga: o so corrige os endereços do processo ao carregar na memória, ou seja, ele soma com o endereço inicial de onde ele está sendo carregado;
* Endereço relativo à base: precisa de um suporte de hardware, pois a CPU tem que ter um registrador de base onde conterá o endereço inicial que será somado a cada instrução executada.

A vantagem desta solução para a solução anterior é que o executável não precisa ter nenhuma informação de controle
de endereço, simplificando a carga. Está soma em hardware não compromete a performase dessa solução, pois o mecanismo de hardware é dedicado a fazer isso, é muito rápido. A CPU Intel tinha um problema que a

distância em relação à base não podia ser muito grande, a limitação era 64Kb, à distância do endereço da base. Se o programa ".EXE" tivesse menos que 64Kb, então, podia usar apenas resolução em tempo de carga. Caso contrário, usava uma solução mista.
* Endereço relativo à instrução corrente: A CPU tem que ter instruções cujo endereço não é endereço que conta a partir do 0, a partir da base, são endereços que contam a partir da própria instrução. Nesta solução não importa onde o programa foi carregado na memória, pois a distância de uma instrução desvio para o seu destino, não se altera. O código que só usa este tipo de endereço é chamado código relocável, porque ele pode ser mudado de posição na memória e continuar executando sem precisar fazer mais nada.
* Segmentação: utiliza uma tabela de segmentos (guarda o início e o tamanho do segmento), dessa forma, visualiza a memória unidimensional de uma forma bidimensional. Uma vantagem é o hardware conhecer as áreas, dessa forma, controla melhor a memória, impede que o processo mexa em áreas que não são suas, ou seja, impede que não haja invasão de segmentos. Outra vantagem, é que não importa para o programa onde está o segmento de memória, pois, o programa sabe que é um endereço bidimensional, a memória é contínua, o programa nem sabe que existe, ele só sabe o número do segmento e o deslocamento. A segmentação gera fragmentação de memória. Para solucionar isso tem duas soluções: evitar que aconteça (com algoritmos de escolha de área livre) e compactação de memória.
Compactação é agrupar em um lado as áreas ocupadas e de outro lado as áreas livres.
A segmentação facilita a compactação. Fazer a compactação, e apenas trocar o endereço inicial do segmento na tabela de
segmento, pois o deslocamento e o tamanho são os mesmos.
* Paginação: Com a paginação passou a existir duas formas de ver a memória, o espaço de endereçamento lógico (páginas virtuais) e o espaço de endereçamento físico (memória dividida em páginas reais). Cada processo tem seu espaço de endereçamento lógico, ou seja, ele pensa que está sozinho e a memória é toda dele. Com a paginação, basta ter memória real livre em qualquer posição que será mapeada na memória lógica.
Praticamente não tem fragmentação, passa a ter somente fragmentação interna na página que é irrelevante.

 # 2 - Quais são os "campos de uma TLB? O que ocorreria se a TLB não existisse? Esta conseqüência é boa ou ruim? Justifique.
Resposta:
A TLB tem os seguintes campos: o endereço lógico, endereço físico e o bit de válido. O bit de válido informa ao hardware que a página não é válida, então, o hardware gera uma interrupção para o SO tratar. TLB é um cache de hardware especial, pequeno e de busca rápida, formado por um grupo de registradores associativos construídos com memória rápida. Cada registrador consiste em duas partes, uma chave e um valor, quando um item é apresentado aos registradores associativos, ele é comparado com todas aschaves ao mesmo tempo. Se o item for encontrado, o campo de valor correspondente será apresentado. A pesquisa é rápida, porém o hardware é caro. Por ser caro o número de entradas é limitado, com isso não sendo possivel carregar toda a tabela nos registradores associativos. Os registradores associativos são usados com as tabelas de página de modo que eles contêm apenas algumas entradas da tabela de página. Quando um endereço lógico é gerado pela CPU, seu número de página é apresentado para um conjunto de registradores associativos que contêm os números de página e seus números de página real correspondentes. Se o número de página lógica for encontrado nos registradores associativos, seu número de página real estará imediatamente disponível e será usado para acessar a memória. Caso o número da página lógica não esteja nos registradores associativos, uma referência de memória à tabela de página deverá ser feita para pegar o número da página real. Esse número de página lógica e o número da página real serão acrescentados aos registradores associativos de modo que possam ser encontrados rapidamente na próxima referência. Se o TLB já estiver cheio de entradas, o sistema operacional deverá escolher uma para substituição. Toda vez que uma nova tabela de página é selecionada (cada troca de contexto), o TLB deve ser apagado (operação de flush) para garantir que o próximo processo a ser executado não use as informações de tradução erradas. Caso contrário, na TLB tem que ter um campo de identificação do processo, se não , haveria entradas antigas no TLB contendo endereços virtuais válidos mas endereços reais incorretos.
A percentagem de vezes em que um número de página é encontrado nos registradores associativos é chamada taxa de acerto. Uma taxa de acerto de 80% significa que encontramos o número de página nos registra dores associativos 80% das vezes. Se não existisse a TLB, a conversão senpre faria dois acessos a memória. Com a TLB é realizada apenas um acesso a memória, com alta taxa de acerto, a conversão é rápida na maioria dos casos. Sem a TLB ficaria muito mais lento.

PF 2002/2

 # 1 - O que aconteceria se não existisse TLB? Por que? TLB é necessária quando existe somente segmentação? Por que?
Resposta:

Sem a TLB haveria uma perda grande de velocidade, porque, toda conversão de endereço lógico para físico deveria acessar a memória, que é onde se encontra a tabela de página. Como o acesso à memória é lento teríamos uma grande perda de velocidade. A TLB NÃO é necessária quando existe segmentação, pois já existe um suporte de hardware que são os registradores de segmentos que contém o número do segmento referente a instrução em execução. Este valor de segmento só é alterado quando houver uma instrução que altere o segmento. Neste momento é que haverá a ida a memória,
para pegar o segmento alterado, na tabela de segmento.

 # 3 - Para que serve o binding de endereços de um programa? Exemplifique os diferentes momentos em que o binding pode ser realizado.
Resposta:
O binding serve para localizar um endereço de um programa em qualquer lugar da memória que este programa venha a ser carregado. O binding pode ser realizado no momento da compilação, em tempo de carga do programa, em tempo de execução do programa, onde neste último precisa de um suporte especial de hardware, registradores ou estruturas mais complexas como a TLB.

 # 4 - O que é Working set? Em quais tipos de alocação de páginas físicas (global ou por processo) é importante o conceito
de Working Set? Por que?
Resposta:
WorkingSet é o número de páginas físicas que o processo está usando em um certo instante de tempo. Na alocação local (por processos) o working set é mais importante, pois se o número de páginas físicas reservadas a um processo for igual ou
maior que o workingset do processo, quase não vai ocorrer falta de página. Só volta a ocorrer falta de página quando trocar o workingset. Não tem como o SO saber o workingset do processo, o que ele faz é observar as faltas de páginas. Se o número de falta de página for grande, o SO sabe que tem que dar mais páginas físicas para o processo. Caso contrário, ele pode até diminuir o número de páginas físicas do processo que não vai influenciar na velocidade de execução. Já na alocação global, o SO não se preocupa com o número de páginas físicas do processo, então, o workingSet não é levado em conta. Mas, mesmo na alocação global, se o processo conseguir o número de páginas que necessita, vai executar e não ficar paginando.