Introdução

1) Quais as 2 principais maneiras de se caracterizar um S.O. e quais suas principais funções?

As duas principais funções do sistema operacional são a de gerenciador de recursos (perspectiva botton-up) e a de máquina estendida (perspectiva top-down).

Na função de gerenciador de recursos, o sistema operacional é responsável por controlar de maneira eficiente os dispositivos de entrada e saída e outros recursos do sistema como a CPU e a memória. Além disto, o sistema operacional é responsável por medir a utilização do sistema e mediar pedidos de recursos de diferentes programas usuários.

Na função de máquina estendida (ou máquina virtual), o sistema operacional abstrai os detalhes mais complexos do hardware, fornecendo ao usuário uma forma mais fácil de programar e usar o sistema computacional. O sistema operacional dispõe uma variedade de serviços que os programas usuários podem obter usando as instruções especiais conhecidas como chamadas de sistema (system calls).

2) Qual a diferença entre modo kernel e modo usuário?

Modo Kernel – Acesso completo a todo o hardware e pode executar qualquer instrução que a máquina seja capaz de executar. O modo kernel impede que serviços e aplicações modo usuário acessem áreas criticas do sistema operacional.

Modo Usuário - Não pode executar instruções que afetam o controle da máquina ou fazem E/S.

Um processo deixa de executar em modo usuário e passa a executar em modo kernel quando executa uma chamada de sistema. Esta diferença é importante porque resulta em segurança e integridade do sistema operacional e do hardware.

3) O que os sistemas operacionais de tempo real tem de diferente dos S.O. comuns?

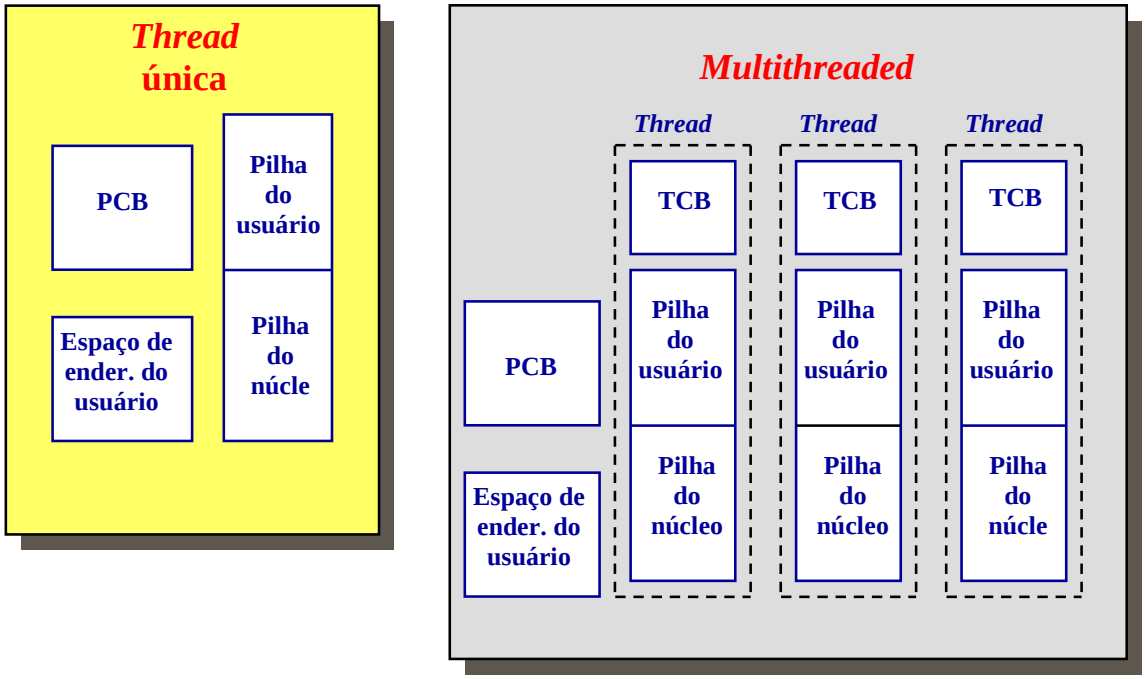
A principal diferença existente entre sistemas operacionais de tempo real (crítico e não crítico) para S.O. comuns é que os primeiros precisam ter uma política de escalonamento baseada em prioridades. Além disso, todos os processos a serem executados pelo sistema operacional precisam ter tempos de execução bem definidos, de modo que as requisições consigam serem atendidas, mesmo no pior caso de execução. Por exemplo, é razoável que num sistema de um carro, o processo de um Freio ABS tenha prioridade maior que a de um vidro elétrico e que, mesmo no pior caso, o Freio ABS possam executar na ordem de, pelo menos, milissegundos.

4) O que são processos e o que são threads?

Podemos dizer que processos são programas em execução. Cada processo é constituído, basicamente, de código executável, dados referentes ao código, pilha de execução, valor do contador do programas (PC), do valor do apontador de pilha (SP), dos valores dos registradores de hardwares envolvidos, além de outras informações necessárias à execução do programa.

Thread, por sua vez, são partes de um dado programa os quais rodam em um pseudoparalelismo. Pseusoparalelismo pois, similar a o que ocorre com processos, existe escalonamento de threads; no entanto, esse é realizado de forma mais rápida do que ocorre com processos, pois estes podem comunicar-se sem invocar o núcleo, visto que compartilham memória e arquivos em comum. (*ver desenho abaixo*)

O principal problema das threads é o não-determinismo junto com sua dependência de velocidade.



5) Quais as vantagens do S.O. ser monolítico ou em camadas?

O sistema operacional monolítico (não dividido em camadas) possui a vantagem de ser menor, geralmente mais rápido. É uma solução geralmente usada quando há restrições de tempo e espaço como em sistema embarcados.

Os sistemas ditos como divido em camadas geralmente oferecem uma maior modularidade o que permite, por exemplo, acoplar dispositivos não previstos durante o desenvolvimento ao sistema posteriormente. Para oferecer tal benefício são divididos em camadas hierarquicamente diferentes. Ou seja, são sistemas que facilitam a evolução e adaptação a novos ambientes.

Esse segundo tipo de sistema operacional geralmente possui um código mais legível.

6) Quais informações contêm e pra que serve o bloco de controle de um processo ou tabela de processo (process control block)?

Podemos dizer que o bloco de controle de cada processo possui as informações essenciais à execução daquele processo. Informações como: valor dos registradores em uso por aquele processo, contador de programa, estado do programa, ponteiro de pilha, prioridade, parâmetros de escalonamento, identificador (ID) do processo, ponteiro para o segmento de código, ponteiro para o segmento de dados como também ponteiro para o segmento de pilha.

A função de armazenar todos esses dados é garantir o funcionamento correto do processo e evitar a perda de dados durante o escalonamento de processos ou tente invadir áreas de memória de outros processos.

7) Um usuário deseja fazer um sistema de *sort* e percebe que usar 10 tarefas que se comunicam para atingir o objetivo é a melhor solução. Ele resolveu testar duas implementações: a) usando 10 processos; b) usando 10 threads de um mesmo processo. Ao executar as duas soluções num ambiente com escalonamento round-robin ele percebeu que uma solução era melhor se o sistema estivesse sobrecarregado (por 100 outros processos, por exemplo) e a outra era melhor se o sistema estivesse sem carga. Qual era a melhor solução para cada caso e porquê? (Dica: a mudança de contexto entre threads leva menos tempo do que entre processos).

Temos, portanto, duas situações:

a.) Sistema sobrecarregado:

Nesse caso é melhor utilizar 10 processos e não 10 threads para executar o sort. Supondo que o sistema é capaz de escalonar 100 processos e na fila de escalonamento já existem 90 processos. Se você colocar 10 processos, seu sort estará ocupando 10% do processamento da CPU; se colocar 10 threads, seu sort estará ocupando somente 1% do processamento da CPU.

b.) Sistema sem carga:

Se o sistema está sem carga (poucos processos na fila de escalonamento) é mais interessante utilizar a abordagem usando 10 threads; pois, o seu único processo irá ocupar quase todo o uso da CPU (em função da baixa carga) e nesse tempo irá escalonar 10 threads que seria como se tivesse 10 partes do seu programa rodando em paralelo.

Utilizar 10 processos não é uma boa idéia, pois o escalonamento de processos é mais lento do que o de threads.

Escalonamento

1) Para que serve a multi-programação?

A multi-programação é uma técnica a qual consiste no rápido chaveamento do processador entre vários programas em execução (processos) conferindo a idéia de que os processos estão rodando em paralelo (pseudoparalelismo).

Essa técnica é utilizada para otimizar o uso do processador; pois, caso um processo esteja sem dados essenciais para prosseguir o seu funcionamento (esperando o usuário digitar uma letra) ele é bloqueado até que esteja pronto para rodar cedendo seu lugar no uso da CPU para outro processo pronto.

Os processos podem estar em 3 estados: pronto, com todos os dados necessários ao seu funcionamento, mas a CPU está ocupada no momento; rodando, sendo executado pelo processador; bloqueado, esperando algum dado faltando a sua execução.

2) O **que são processos I/O-bound e CPU-bound?**

Processos I/O-bound são aqueles os quais durante sua execução requisitam muitos serviços de entrada/saída (input/output).

CPU-bound, por sua vez, são aqueles processos os quais durante sua execução não requisitam serviços de entrada/saída; consomem somente muito processamento de dados pela CPU.

Em muitos sistemas, como UNIX, a prioridade no escalonador é estabelecida colocando processos I/O-bound como prioritários. Por que? Estes solicitam muita Entrada/Saída e provavelmente ficaria ociosos rodando no processador. Por isso, sua entrada em processamento é liberada, para que ele possa voltar ao estado de Bloqueado (E depois, Pronto), deixando o processador livre para os processo CPU-bound.

3) O que são processos *batch* e interativos?

Processos batch são aqueles os quais apresentam pouca interatividade com o usuário. Normalmente são CPU-bound mas podem ser I/O-bound, por exemplo: o desfragmentador de disco (não interage com o usuário durante sua execução, mas está requisitando constantemente serviços de entrada/saída; no caso, acesso ao disco).

Processos interativos, como o próprio nome sugere, são aqueles os quais geralmente passam pouco tempo no estado pronto e boa parte do tempo no estado bloqueado (esperando dados a serem fornecidos pelo usuário).

4) Defina: fairness, starvation, turnaround, throughput.

Fairness: palavra do inglês cujo tradução é justiça. No nosso contexto, significa garantir que todos os processos do sistema terão chances iguais (política justa) de usar o processador.

Starvation: palavra a qual pode ser traduzida (no nosso contexto) como estagnação. Por exemplo, escalonamentos com políticas dotadas de

prioridade (os processos possuem prioridades) pode gerar uma situação na qual um certo processo nunca será executado. Esse fenômeno é denominado starvation. Um caso onde isso pode acontecer facilmente é no escalonamento Híbrido, baseado em filas múltiplas, onde um processo pode ir parar numa fila de prioridade baixíssima e não ser executado.

Turnaround: é a política cujo objetivo é minimzar o tempo com que os processos batch precisam para serem executados completamente. Ou seja, o tempo que programas batch devem esperar para gerar a saída.

Throughput: é a política cujo intuito é maximizar o número de “jobs” (trabalhos) processados por unidade de tempo.

5) Defina região crítica e quais são as condições necessárias para garantir exclusão mútua.

Uma região crítica é uma área de código de um algoritmo que acessa um recurso compartilhado que não pode ser acessado concorrentemente por mais de uma linha de execução.
As 4 condições necessárias são:
- Nunca dois processos estarão simultaneamente em uma região crítica
– Não se pode considerar velocidades ou números de CPUs
– Nenhum processo executando fora de sua região crítica pode bloquear outros processos
– Nenhum processo deve esperar eternamente para entrar em sua região crítica

6) Defina e descreva o processo de interrupção. Qual a diferença entre interrupções síncronas e assíncronas?

Uma interrupção é um sinal de um dispositivo que tipicamente resulta em uma troca de contexto, isto é, o processador pára de fazer o que está fazendo para atender o dispositivo que pediu a interrupção. Quando ocorre uma interrupção, a CPU interrompe o processamento do programa em execução e executa um pedaço de código (tipicamente parte do sistema operacional) chamado de tratador de interrupção. Após a execução do tratador, a CPU volta a executar o programa interrompido

Interrupções Assíncronas - geradas por algum dispositivo externo à CPU, ocorrem independentemente das instruções que a CPU está executando, não há qualquer comunicação entre o programa interrompido e o tratador. Ex: interrupção de relógio, interrupção de I/O.

Interrupções Síncronas – Também chamadas de Traps, são geradas pelo programa em execução, em consequência da instrução sendo executada, algumas são geradas pelo hardware em situações em que o programa não teria como prosseguir, ocorrem em função da instrução que está sendo executada, logo o programa passa algum parâmetro para o tratador.
Ex: overflow

7) Qual a diferença entre escalonamento preemptivo e não-preemptivo?

Os preemptivos são algoritmos que permitem que um processo seja interrompido durante sua execução, quer seja por força de uma interrupção de entrada/saída, quer seja em decorrência da politica de escalonamento

adotada e aplicada por parte do escalonador de processos ou simplesmente por força do término da execução do processo.

Nos algoritmos não-preemptivos, esse fato não ocorre, sendo cada programa executado até o fim, ou seja, sem interupções.

8) Qual a vantagem de ter um quantum pequeno ou um quantum grande?

Se for definido um quantum pequeno, o tempo de resposta para processos interativos é diminuído. É uma técnica que permite uma sensação de multiprogramação maior. Mas apresenta problemas como: se o quantum for igual ao tempo necessário para a mudança de contexto, perde-se metade do tempo do processador somente com mudanças de contexto.

Se for definido um quantum grande, as mudanças de contexto entre processos é diminuído e, conseqüentemente, diminui-se também o overhead do sistema operacional. Também apresenta problemas: se for definido um quantum muito grande, os processos interativos terão seu tempo de resposta aumentado consideravelmente.

9) Explique como funciona o escalonamento round-robin e quando deve ser usado.

O escalonamento round-robin é baseado em uma lista sem prioridade, baseado em um sistema preemptivo.

Cada processo é colocado em uma fila de processos e para cada um é destinado um tempo máximo (quantum) durante o qual aquele processo poderá usar o processador. Se após esse período de tempo o processo ainda estiver usando o procesador, ele irá para o estado pronto e outro processo será executado pela CPU. Esse troca de contexto pelo sistema operacional (baseado em interrupções) é o que caracteriza um sistema preemptivo. Se um processo for bloqueado ou terminado, antes do fim de seu quantum, a comutação dar-se-á no exato momento do término ou bloqueio, fazendo com que o processador não fique ocioso.

É uma política simples e justa. Deve ser usada em sistemas interativos (com muita interação com o usuário).

10) Explique como funciona o escalonamento FIFO e quando deve ser usado.

O escalonamento FIFO usa uma lista de processos sem prioridade (semelhante ao round-robin) mas não é preemptivo; ou seja, não existe a noção de quantum e, conseqüentemente, não permite a suspensão temporária dos processos pelo S.O. fazendo com que cada processo seja executado até seu final. É um algoritmo simples. Bom para sistemas batch. Inviabiliza os sistemas interativos.

11) Explique como funcionam os escalonamentos *shortest job first* e *shortest remaining job next.*

SJF **-** escalonador SJF funciona a partir de um conceito bem simples: os processos menores terão prioridade, ou seja, serão executados primeiro. Isso tem como resultado um tempo médio mínimo de espera para cada conjunto de processos a serem executados. `um tipo de escalonamento

não-preemptivo. Simples e minimiza a quantidade média de tempo de espera de cada processo, porém tem grande potencial de starvation.

SRJN - variante preemptiva do escalonamento SJF. A fila de processos a serem executados pelo SRJN é organizada conforme o tempo estimado de execução, ou seja, de foma semelhante ao SJF, sendo processados primeiros os menores jobs. Na entrada de um novo processo, o algoritmo de escalonamento avalia seu tempo de execução incluindo o job em execução, caso a estimativa de seu tempo de execução seja menor que o do processo concorrentemente em execução, ocorre a substituição do processo em execução pelo recém chegado, de duração mais curta, ou seja, ocorre a preempção do processo em execução. Possui alto throughput, porém processos longos podem sofrer com tempo de spera e starvation.

12) Existe algum escalonamento que agrega as características do round-robin e do FIFO ao mesmo tempo? Como funciona?

Escalonamento com prioridade funciona da seguinte maneira: a cada processo é associado uma prioridade, e o processo pronto com maior prioridade será aquele que vai rodar primeiro. Para evitar que processos com alta prioridade monopolizem o processador, o escalonador decrementa a prioridade do processo que está rodando, a cada interrupção de tempo, até que a prioridade do processo corrente torne-se mais baixa que a do de mais alta prioridade da fila de “pronto”, trocando-se assim o contexto.

R: A resposta mais precisa é **SIM**. Como funciona segue no quesito abaixo, o **9**.

13) Explique o escalonamento de filas múltiplas (Multiple Feedback Queue).

O algoritmo MFQ (Multiple Feedback Queue) é um tipo de escalonamento híbrido, ou seja, possui características do round-robin como também do FIFO.

É um escalonamento de processos com prioridade. Sua estrutura é composta por várias filas de processos sendo a primeira com maior prioridade (menor quantum) e a última com menor (maior quantum). Em cada fila tem-se o uso do round-robin para escalonar os processos.

O MFQ funciona da seguinte forma: ao ser criado um novo processo, ele é alocado na primeira fila (prioridade mais alta e quantum menor); em seguida, ao ser executado (respeitando a prioridade das filas), cada processo poderá descer ou subir uma fila. Irá subir se ele fizer uma requisição de I/O antes de acabar seu quantum, irá descer se ainda estiver sendo executado ao acabar seu quantum.

Esse algoritmo faz com que se acumulem na primeira fila os processos I/O-bound (a maioria deles no estado bloqueado) e na última fila os CPU-bound, fila a qual contém os processos que passam maior tempo sendo executados pela CPU.

14) **Explique o escalonamento de Uso de Partições de Lote**

O sistema aceita tantos processos batch quantas forem as partições de lote, em seguida, o sistema aceita todos os processos interativos. O escalonamento ocorre em dois níveis. Existe separação entre os processos

interativos e os processos batch de forma que processos interativos são ativados imediatamente e processos batch esperam a liberação do lote

15) O que é um sistema *time-sharing*?

Como a própria tradução sugere, sistemas time-sharing são aqueles nos quais existem compartilhamento de tempo de CPU entre os processos. Essa política tenta evitar que um dado processo monopolize o sistema.

16) Podemos ter um sistema *time-sharing* sem usar interrupções?

Não. A interrupção do relógio (por exemplo) é o dispositivo responsável por indicar ao sistema operacional que um dado processo em execução ultrapassou seu quantum e que é necessário levá-lo para o estado pronto e colocar outro processo para executar. Logo, um sistema sem interrupções não teria como detectar se o processo ultrapassou ou não seu quantum.

Gerenciamento de Memória

1) Quais as vantagens de ter memória virtual?

A principal vantagem de se ter memória virtual é basicamente simular para o programador (programa) a existência de uma memória tão grande seja sua necessidade.

Os programas os quais são maiores que a quantidade de memória disponível são divididos em módulos (overlays) os quais cabem na memória principal ficando o resto no disco. É função do sistema operacional manter na memória principal as partes do programa efetivamente em uso, deixando o restante no disco.

A memória virtual, portanto, facilita a vida do programador que não fica restrito à quantidade real de memória existente no computador. Nem precisar exercer a atividade estafante de mapear endereços virtuais em endereços reais.

2) O que o princípio da localidade tem a ver com a hierarquia de memória?

O princípio da localidade possui dois tipos: temporal, afirma que uma dada instrução referenciada no momento provavelmente será referenciada de novo em breve; espacial, afirma que as instruções próximas a instrução referenciada no momento serão provavelmente referenciadas em breve.

Baseado nesse princípio, pode-se ter uma idéia de que partes do programa (processos) serão mais referenciados logo em seguida. Logo, utilizando essa previsão com a hierarquia de memória. Costuma-se colocar os processos com maiores chances de serem referenciados em breve em uma memória mais rápida (cache, por exemplo); os que serão provavelmente referenciados não tão breve em uma memória não tão rápida (memória principal) e os que não serão referenciados logo, em uma memória mais lenta (memória secundária).

3) Para que a relocação de código em memória é usada?

A relocação de código em memória é usado para permitir que os programas executem corretamente em qualquer lugar da memória. Ou seja, quando o endereço de memória no qual o programa será carregado não é conhecido em tempo de compilação.

Seu funcionamento se dá da seguinte forma: o programa é alocado em uma dada posição da memória e ao ser executado o S.O. recalcula os endereços relativos presentes no código em função de onde o programa foi carregado na memória obtendo o endereço físico real. Essa operação é feita utilizando a tabela de relocação a qual contém para cada entrada um endereço base a ser somado ao endereço relativo do programa obtendo o endereço correto do programa na memória.

4) Quais as técnicas que você conhece para fazer relocação de código em memória? Explique cada uma delas.

Existem basicamente três técnicas. Nos primórdios da computação, era função do programador informar no início do programa qual era o endereço base no qual o programa seria carregado na memória.

Posteriormente, virou função do S.O. descobrir qual era o endereço real da memória na qual o programa estava em função de uma tabela de símbolos gerada pelo compilador.

Atualmente, para a relocação de código em memória ser mais eficiente (rápida) utiliza-se um hardware específico: registradores que armazenam o endereço base no qual o programa foi carregado como também o endereço limite daquele programa.

5) Como funciona um sistema de multiprocessamento com partição fixa? Como ocorre a fragmentação neste sistema?

Em um sistema de multiprocessamento com partição fixa a memória é dividida em um número de partições com tamanhos pré-estabelecidos (fixo). Quando um processo é iniciado, ele é colocado numa fila de entrada para ocupar a menor partição suficiente para acomodá-lo. Como as partições possuem tamanho fixo, é comum haver espaços em uma dada partição não utilizados pelo processo.

Esse espaço de memória é desperdiçado, caracterizando uma fragmentação interna.

6) Como funciona um sistema de multiprocessamento com partição variável? Como ocorre a fragmentação neste sistema?

Em um sistema de multiprocessamento com partição variável, à medida que os processos são criados, a memória aloca exatamente o tamanho necessário para a execução daquele processo, como se eles tivessem tamanhos fixos. Deve-se lembrar, que neste esquema, cada processo tem um espaço alocado da seguinte maneira: área de código do processo, área de dados do processo, espaço reservado ao crescimento da área de código ou de pilha do processo, e área de pilha.

Quando um processo sai da memória, aquele espaço antes ocupado fica livre. Se o próximo processo a ser alocado for menor que aquele espaço, ele é alocado ali.

Com o tempo, começam a aparecer espaços não alocados entre as partições de tamanhos variados, caracterizando uma fragmentação externa a qual pode ser contornada usando compactação com realocação.

7) Para que serve a paginação? Como funciona? Como ocorre a fragmentação neste sistema?

A paginação é a técnica usada para traduzir, utilizando uma tabela de páginas, um endereço virtual (memória virtual) em um endereço real.

Os endereços de memória gerados pelo programa são chamados de endereços virtuais e formam o espaço de endereçamento virtual o qual é dividido em páginas. Esse endereço virtual é enviado para a MMU (unidade de gerenciamento de memória) a qual, utilizando a tabela de páginas, transforma o endereço virtual em um endereço real, indicando em qual moldura da memória real essa página está referenciada.

A fragmentação nesse sistema ocorre na última página. Pois o processo pode vir a não ocupar a última página completamente. Fragmentação interna, no caso.

8) Para que serve a segmentação? Como funciona?

A segmentação é utilizada para permitir que partes de um processo (código, dados, pilha, etc.) possam ser quebrados em espaços de endereçamento logicamente independentes o que auxilia nas questões envolvendo compartilhamento e proteção de dados.

São criados segmentos independentes. Um segmento para dados, um segmento para código e assim por diante. Cada segmento começa do endereço lógico zero e pode crescer dinâmicamente (até um certo limite suficientemente grande) sem se preocupar em invadir o espaço de código (por exemplo).

Para identificar o início de cada segmento utiliza-se um registrador específico (por exemplo, data segment para o segmento de dados) cujo valor precisa conter o endereço real do início daquele segmento na memória. Essa função de indicar esse endereço é função do programador.

O programa, ao acessar um endereço relstivo naquele segmento, esse valor é somado com o registrador base do segmento indicando a posição real na memória.

9) Compare paginação com segmentação.

Na paginação, o programador não interage com a técnica, só um espaço de endereços lineares é usado, o espaço de endereçamento pode ser maior do que a memória física, as tabelas não mudam de tamanho, os procedimentos e dados estão juntos e não é fácil compartilhar procedimentos.

Já na segmentação, o programador interage com a técnica, vários espaços de endereçamentos lineares são usados, o espaço de endereçamento também pode ser maior do que a memória física, as tabelas podem mudar de tamanho, os procedimentos e dados são alocados separadamente e protegidos individualmente (tal política facilita também o compartilhamento deles entre processos diferentes).

10) Que técnicas podem ser usadas para reduzir o tempo de acesso e o tamanho das tabelas de página num sistema de paginação de memória? Descreva cada técnica mencionada.

A principal técnica utilizada para reduzir o tempo de acesso da tabela de página em um sistema de paginação de memória é o TLB (solução em hardware).

As técnicas que podem ser utilizadas para diminuir o tamanho das tabelas de página em um sistema de paginação de memória são o uso de tabelas de página invertida e também da tabela de páginas multi-nível.

A explicação de cada uma está contida na resposta das próximas questões.

11) Como funciona a tabela de páginas comum num sistema de paginação de memória?

A tabela de páginas contém uma página para cada endereço virtual possivelmente referenciado. Cada página contém informações como: se aquela página foi modificada, referenciada, se é válida e o endereço real na memória.

Para obter o endereço real de uma página virtual, confere-se se aquela página virtual é válida. Se for, pega-se o endereço real e soma-se ao deslocamento da página virtual; se não for, detectou-se uma falta de página.

A desvantagem dessa implementação é que geralmente as tabelas são grandes e precisam de um grande espaço de memória para serem armazenadas.

12) Como funciona a tabela de páginas multi-nível num sistema de paginação de memória?

A tabela de páginas multi-nível é bastante similar a tabela de páginas comum. A grande diferença consiste em se ter uma tabela onde cada célula aponta para uma outra tabela onde a tabela do último nível é a que contém o endereço real daquela página virtual.

Essa abordagem reduz o tamanho da tabela uma vez que nem todas as tabelas de cada nível precisam estar na memória principal. Algumas podem estar armazenadas em disco. Essa política diminui o espaço de memória necessário para armazenar a tabela de páginas.

13) Como funciona a tabela de páginas invertida num sistema de paginação de memória?

Essa é outra implementação visando diminuir o tamanho da tabela de páginas. Cada página virtual é mapeada em uma posição de uma tabela menor por uma função hash a qual gera um certo hash code. Se naquela posição a página virtual for equivalente a página virtual em mão, então o endereço real daquela posição é o qual nos interessa. Se nao for, é porque ocorreu uma colisão. Para saber em qual posição tentou-se inserir a página colidida, existe um link apontando para essa outra posição.

Essa implementação pode se tornar custosa se houver um número elevado de colisões em função de uma função hash ruim (distribui de forma não uniforme as entradas).

14) Como funciona o Translation Lookaside Buffer e pra que serve num sistema de paginação de memória?

O TLB é uma tabela (implementada em hardware) localizada na MMU a qual contém as páginas virtuais, com seus respectivos endereços reais, mais referenciadas recentemente.

Funciona como uma tabela de páginas comum, mas por ser implementada em hardware utilizando uma memória associativa (todas as entradas são comparadas em paralelo com a página virtual a qual se está investigando o endereço real) é consideravelmente mais rápida.

15) Qual o melhor algoritmo realista de substituição de páginas num sistema de paginação de memória?

Dentre os vários algoritmos de substituição de páginas em um sistema de paginação de memória, o mais realista é o chamado algoritmo do relógio. O WSClock também constitui um algoritmo de substituição eficiente.

16) Para que serve o Modelo de Conjunto de Trabalho (ou pré-paginação) num sistema de paginação de memória?

Conjunto de trabalho é o conjunto de páginas que um processo está atualmente usando. Portanto, o modelo do CT (conjunto de trabalho) é a forma de tentar manter controle sobre o CT de processos, de forma a garantir que um deles esteja na memória durante a execução do processo que lhe é correspondente.

O objetivo dessa política é reduzir de forma significativa a ocorrência de falta de páginas(thrashing), carregando a página antes mesmo do processo iniciar sua execução.

17) Que implicações podemos esperar se tivermos páginas muito pequenas ou muito grandes num sistema de paginação de memória?

Como dito anteriormente, em média, metade da última página de um processo estará vazia, ou seja, ter-se-á memória desperdiçada por fragmentação interna. Logo, quanto maior for o tamanho das páginas, maior será a quantidade de bytes perdidos.

Por outro lado, páginas pequenas em demasia significam que os programas irão precisar de muitas páginas, resultando em uma tabela de páginas muito grande.

Ou seja, o tamanho da tabela de páginas é grande quando o tamanho da página é pequeno; a fragmentação é grande quando o tamanho da página for grande.

Deseja-se, portanto, atingir-se um equilíbrio no qual as páginas nem sejam nem muito grandes nem muito pequenas. Prova-se matematicamente que isso ocorre quando p = raiz(2*s*e).

Obs.: s = tamanho médio de um processo;
P = tamanho da página;
E = tamanho da entrada na tabela.

18) Que implicações podemos esperar se tivermos uma política local ou uma política global de alocação de páginas para processos num sistema de paginação de memória?

Na política local são atribuídas a cada processo um número fixo de molduras de páginas, já os processos globais alocam dinamicamente as molduras de páginas aos processos. Se o tamanho do conjunto de trabalho for variável, a política global funciona melhor pois na local se o conjunto do trabalho do processo vier a crescer, o resultado será a ocorrência de trashing, mesmo que haja muitas molduras livres na memória principal. Já se o tamanho na CTP vier a diminuir, o resultado será um grande desperdício de memória.

O descrito no parágrafo anterior ocorre pois na política local quando ocorre falta de páginas quando um processo estiver executando, a página que irá ser escolhida para remoção pertence a esse processo. Se esse processo tiver poucas páginas, a retirada de uma dessas poucas páginas poderá causar novamente uma falta de página.

Na políta global, a página a ser retirada é escolhida no escopo de todas as páginas que estiverem na memória, mesmo as de outros processos.

Deadlocks

1) O que é deadlock e como ele ocorre?

Um conjunto de processos está em situação de deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer.

Ocorre quando um processo X que detem um recurso A solicita um recurso B do processo Y e ao mesmo tempo o processo Y solicita o recurso B do processo A. Ambos são bloqueados e assim permanecem

2) Que estratégias podemos utilizar para o tratamento de deadlocks?

Podemos utilizar 4 técnicas:

- Algoritmo do Avestruz: finge que o problema não existe. É razoável se deadlocks são raros. Utilizado por UNIX e Windows.

- Recuperação: Existem 3 tipos de recuperação.
  - Através de preempsão: retira um recurso de outro processo
  - Através de reversão de estado: monitora um processo e se ele entra em deadlock é reiniciado
  - Através de eliminação de processos: forma mais simples, simplesmente elimina um dos processos e assim os outros processos recebem seus recursos

- Prevenção: Existem 4 tipos de prevenção:
  - Exclusão Mútua: utilização de spool. Ex: impressoras
  - Posse e Espera: exigir que todos os processos requisitem os recursos antes de iniciarem. Assim processos podem não saber que recursos precisam no início da execução ou podem reter recursos que poderiam ser usados por outros processos.
  - Não preempção: retornam-se os recursos alocados
  - Espera Circular: ordenam-se numericamente os processos

- Evitação Dinâmica: Alocam-se cuidadosamente os recursos.