INSTITUTO FEDERAL DO ESPÍRITO SANTO

Sistemas Operacionais

CAMPUS SERRA

Caderno de Exercícios

Prof. Flávio Giraldeli

v1.0.0

Sumário

Notas

- Os exercícios estão classificados segundo o grau de aprendizagem adquirida. Apesar de ser recomendado que todos os exercícios sejam feitos, deve-se dar prioridade aos com mais asteriscos, ou seja...

  [***] > **[**]** > **[*]** > "nada"

  Quanto mais asteriscos, mais "interessantes" eles foram considerados pelo professor. ☺
- Os exercícios puramente conceituais (aqueles cuja resposta pode ser facilmente encontrada no livro texto e/ou slides) são importantes para revisão e estão marcados com **[R]**. Atenção: Isso **não** os torna desprezíveis!

Cap. 1: INTRODUÇÃO

1) [R] Quais são as três finalidades principais de um sistema operacional?

Resposta:

As três finalidades principais são:

- Fornecer um ambiente para que um usuário de computador execute programas no hardware do computador de maneira conveniente e eficiente.
- Alocar os recursos separados do computador conforme é preciso para resolver o problema dado. O processo de alocação deve ser o mais claro e eficiente possível.
- Como programa de controle, ele serve a duas funções principais: (1) supervisão da execução dos programas de usuário para evitar erros e o uso impróprio do computador e (2) gerenciamento da operação e controle dos dispositivos de I/O.

2) [*] Em um ambiente de multiprogramação e tempo compartilhado, vários usuários compartilham o sistema simultaneamente. Essa situação pode resultar em diversos problemas de segurança.

a. Cite dois desses problemas.

b. Podemos assegurar o mesmo nível de segurança tanto em uma máquina dedicada como em uma máquina de tempo compartilhado? Explique sua resposta.

Resposta:

a. O roubo e a cópia de programas ou dados de alguém; a utilização de recursos do sistema (CPU, memória, espaço em disco, periféricos) sem a responsabilização necessária.

b. Provavelmente não, já que qualquer esquema de proteção imaginado por humanos poderá ser inevitavelmente quebrado por um humano, e quanto mais complexo o esquema mais difícil é se sentir confiante em sua implementação correta.

3) A questão da utilização de recursos assume formas distintas em diferentes tipos de sistemas operacionais. Liste que recursos devem ser gerenciados cuidadosamente nas configurações a seguir:

a. Sistemas mainframe ou de minicomputador

b. Estações de trabalho conectadas a servidores

c. Computadores móveis

Resposta:

a. Mainframes: recursos de memória e de CPU, armazenamento, largura de banda de rede

b. Estações de trabalho: recursos de memória e de CPU

c. Computadores móveis: consumo de energia, recursos de memória

4) [R] Descreva as diferenças entre os multiprocessamentos simétrico e assimétrico. Cite três vantagens e uma desvantagem de sistemas multiprocessadores.

Resposta:

O multiprocessamento simétrico trata todos os processadores como iguais e o I/O pode ser processado em qualquer CPU. O multiprocessamento assimétrico tem uma CPU mestre e as CPUs restantes são escravas. A CPU mestre distribui as tarefas entre as escravas e o I/O é, usualmente, feito somente pela CPU mestre. Multiprocessadores podem economizar dinheiro, mas não podem duplicar suprimentos de energia,

instalações e periféricos. Eles podem executar programas mais rapidamente e podem ter aumento da confiabilidade. Eles também são mais complexos, tanto em hardware como em software, do que os sistemas com um único processador.

5) [*] Defina as propriedades essenciais dos tipos de sistema operacional a seguir:
   a. Batch
   b. Interativo
   c. De tempo compartilhado
   d. De tempo real
   e. De rede
   f. Paralelo
   g. Distribuído
   h. Em Cluster
   i. Móvel

   Resposta:

   a. **Batch**. Jobs com necessidades semelhantes fazem parte do mesmo lote e são executados em um computador como um grupo, por um operador ou um sequenciador automático de jobs.
   b. **Interativo**. Este sistema é composto por muitas transações curtas em que os resultados da próxima transação podem ser imprevisíveis. O tempo de resposta precisa ser curto (segundos), já que o usuário submete e espera pelo resultado.
   c. **De tempo compartilhado**. Este sistema utiliza o scheduling da CPU e a multiprogramação para fornecer o uso interativo econômico de um sistema. A CPU muda rapidamente de um usuário a outro. Em vez de ter um job definido por imagens de cartão em spool, cada programa lê seu próximo cartão de controle do terminal, e a saída é normalmente exibida na tela imediatamente.
   d. **De tempo real**. Utilizado com frequência em uma aplicação dedicada, esse sistema lê informações a partir de sensores e deve responder dentro de um período de tempo fixado para garantir o desempenho correto.
   e. **De rede**. Fornece serviços do sistema operacional, tal como o compartilhamento de arquivos, através de uma rede.
   f. **SMP**. Utilizado em sistemas em que existem múltiplas CPUs, cada uma delas executando a mesma cópia do sistema operacional. A comunicação é feita através do bus do sistema.
   g. **Distribuído**. Este sistema distribui a computação entre diversos processadores físicos. Os processadores não compartilham memória ou um relógio. Em vez disso, cada processador tem sua própria memória local. Eles se comunicam uns com os outros por meio de várias linhas de comunicação, tais como um bus de alta velocidade ou uma rede local.
   h. **Em cluster**. Um sistema em cluster combina múltiplos computadores em um único sistema para executar tarefas computacionais distribuídas através do cluster.
   i. **Móvel**. Um sistema de computação pequeno que executa tarefas simples tais como calendários, email e navegação na web. Os sistemas móveis diferem dos sistemas desktop tradicionais com memória e telas de display menores e processadores mais lentos.

6) [R] Quais são as cinco principais atividades de um sistema operacional relacionadas ao **gerenciamento de processos**?

   Resposta:

As cinco atividades principais são:

a. A criação e destruição tanto de processos de usuário quanto de sistema.
b. A suspensão e retomada de processos.
c. O fornecimento de mecanismos para a sincronização de processos.
d. O fornecimento de mecanismos para a comunicação entre processos.
e. O fornecimento de mecanismos para manipulação de Deadlocks (impasses)

7) [R] Quais são as três principais atividades de um sistema operacional relacionadas ao **gerenciamento de memória**?

Resposta:

As três atividades principais são:

a. Controlar as partes da memória que estão sendo correntemente utilizadas e quem as está utilizando.
b. Decidir que processos devem ser carregados na memória quando o espaço em memória se torna disponível.
c. Alocar e desalocar espaço da memória quando necessário.

8) [R] Quais são as três principais atividades de um sistema operacional relacionadas ao **gerenciamento de memória secundária**?

Resposta:

As três atividades principais são:

a. Gerenciamento do espaço livre
b. Alocação de espaço de armazenamento
c. Scheduling de disco

Cap. 2: ESTRUTURAS DO SISTEMA OPERACIONAL

1) [R] Qual é a finalidade das chamadas de sistema?

Resposta:

As chamadas de sistema permitem que processos de nível de usuário solicitem serviços do sistema operacional.

2) [R] Qual é a finalidade dos programas de sistema?

Resposta:

Os programas de sistema podem ser imaginados como feixes de chamadas de sistema úteis. Eles fornecem funcionalidade básica para usuários de modo que os usuários não precisem escrever seus próprios programas para resolver problemas comuns.

3) [R] Qual é a principal vantagem da abordagem em camadas para o projeto de sistemas? Quais são as desvantagens do uso da abordagem em camadas?

Resposta:

Como em todos os casos de projeto modular, o projeto de um sistema operacional de forma modular tem diversas vantagens. O sistema é mais fácil de depurar e modificar porque as mudanças afetam apenas seções limitadas do sistema em vez de mexer com todas as seções do sistema operacional. As informações são mantidas apenas onde são necessárias e são acessíveis somente dentro de uma área definida e restrita, de modo que quaisquer bugs que afetem os dados devem ficar limitados a um módulo específico ou camada.

4) [**] Liste cinco serviços fornecidos por um sistema operacional e explique por que cada um deles é conveniente para os usuários. Em que casos seria impossível programas de nível de usuário fornecerem esses serviços? Explique sua resposta.

Resposta:

Os cinco serviços são:

a. **Execução de programas**. O sistema operacional carrega o conteúdo (ou seções) de um arquivo em memória e inicia sua execução. Um programa de nível de usuário poderia não ser confiável para alocar tempo de CPU apropriadamente.

b. **Operações de I/O**. Discos, fitas, linhas seriais e outros dispositivos têm que se comunicar em um nível muito baixo. O usuário precisa apenas especificar o dispositivo e a operação a ser executada sobre ele, enquanto o sistema converte a solicitação em comandos específicos do dispositivo ou do controlador. Programas de nível de usuário não podem ser encarregados de acessar somente os dispositivos aos quais eles precisam ter acesso e acessá-los somente quando eles não estiverem sendo usados.

c. **Manipulação do sistema de arquivos**. Existem muitos detalhes na criação, remoção, alocação e nomeação de arquivos que os usuários não devem executar. Blocos de espaço em disco são utilizados por arquivos e devem receber trilhas. A exclusão de um arquivo requer a remoção das informações do arquivo de nomes e a liberação dos blocos alocados. As proteções também precisam ser verificadas para garantir o acesso apropriado ao arquivo. Programas de usuário podem não garantir a adesão aos métodos de proteção e não são confiáveis para alocar apenas blocos livres e desalocar blocos na exclusão do arquivo.

d. **Comunicações**. A passagem de mensagens entre sistemas requer que as mensagens sejam enfeixadas em pacotes de informação, enviadas ao controlador da rede, transmitidas através de um meio de comunicação e remontadas pelo sistema de destino. A ordenação dos pacotes e a correção dos dados precisam ter lugar. Mais uma vez, programas de usuário não podem coordenar o acesso ao dispositivo de rede nem receber pacotes destinados a outros processos.

e. **Detecção de erros**. A detecção de erros acontece em nível tanto de hardware quanto de software. No nível de hardware, todas as transferências de dados devem ser inspecionadas para garantir que os dados não foram corrompidos em trânsito. Todos os dados em mídia devem ser verificados para ter certeza de que eles não mudaram desde que foram gravados na mídia. No nível de software, as mídias devem ser verificadas quanto à consistência dos dados; por exemplo, se o número de blocos de armazenamento alocados e não alocados coincide com o número total do dispositivo. Aqui, os erros são, com frequência, independentes de processo (por exemplo, a corrupção de dados em um disco), de modo que deve existir um programa global (o sistema operacional) que manipule todos os tipos de erro. Além disso, tendo os erros processados pelo sistema operacional, os processos não precisam conter código para capturar e corrigir todos os erros possíveis em um sistema.

5) [R] Como seria o projeto de um sistema que permitisse a escolha do sistema operacional a partir do qual se dará a inicialização? O que o programa bootstrap teria que fazer?

Resposta:

Considere um sistema que quisesse operar tanto o Windows XP como três diferentes distribuições do Linux (isto é, RedHat, Debian e Mandrake). Cada sistema operacional será armazenado em disco. Durante a inicialização do sistema, um programa especial (que chamaremos de gerenciador de inicialização) determinará qual sistema operacional deverá ser inicializado. Isso significa que, em vez de inicializar a princípio um sistema operacional, o gerenciador de inicialização executará primeiro durante o início do sistema. É o gerenciador de inicialização que é o responsável por determinar qual sistema deve ser inicializado. Normalmente, os gerenciadores de inicialização devem ser armazenados em determinadas localizações do disco rígido para serem reconhecidos durante o início do sistema. Gerenciadores de inicialização colocam, com frequência, à disposição do usuário, uma seleção de sistemas para serem inicializados; eles também são normalmente projetados para inicializar um sistema operacional default se nenhuma escolha for feita pelo usuário.

6) [R] Por que a separação entre mecanismo e política é desejável?

Resposta:

Mecanismo e política devem ser separados para garantir que os sistemas sejam fáceis de modificar. Duas instalações de sistemas não são iguais, e, assim, cada instalação deve querer ajustar o sistema operacional às suas necessidades. Com mecanismo e política separados, a política pode ser alterada, enquanto o mecanismo permanece inalterado. Esse arranjo fornece um sistema mais flexível.

7) [R] Qual é a principal vantagem da abordagem de microkernel para o projeto de sistemas? Como os programas de usuário e serviços do sistema interagem em uma arquitetura de microkernel? Quais são as desvantagens do uso da abordagem de microkernel?

Resposta:

Os benefícios normalmente incluem o seguinte: (a) a adição de um novo serviço não requer a modificação do kernel, (b) é mais seguro quando mais operações são realizadas em modalidade de usuário do que em modalidade de kernel, e (c) um projeto de kernel e funcionalidade mais simples normalmente resultam em um sistema operacional mais confiável. Os programas de usuário e os serviços do sistema interagem em uma arquitetura de microkernel utilizando mecanismos de comunicação interprocessos, tais como a emissão de mensagens. Essas mensagens são transmitidas pelo sistema operacional. As principais desvantagens da

arquitetura de microkernel são os overheads associados à comunicação interprocessos e ao uso frequente das funções de transmissão de mensagens do sistema operacional para habilitar o processo do usuário e o serviço do sistema a interagirem um com o outro.

8) [R] Qual é a principal vantagem de usar uma arquitetura de máquina virtual para um projetista de sistema operacional? Qual é a principal vantagem para o usuário?

Resposta:

O sistema é fácil de depurar, e problemas de segurança são fáceis de resolver. As máquinas virtuais também fornecem uma boa plataforma para a pesquisa em sistemas operacionais, já que muitos sistemas operacionais diferentes podem executar em um sistema físico.

Cap. 3: PROCESSOS

1) [**] O Palm OS não fornece um meio de processamento concorrente.

   Discuta três grandes complicações que o processamento concorrente adiciona a um sistema operacional.

   Resposta:

   a. Deve ser implementado um método de compartilhamento de tempo para permitir que cada um dos diversos processos tenha acesso ao sistema. Esse método envolve a preempção de processos que não abandonem voluntariamente a CPU (utilizando uma chamada de sistema, por exemplo), sendo o kernel reentrante (e, portanto, mais de um processo pode estar executando código do kernel concorrentemente).
   b. Os processos e os recursos do sistema devem possuir proteções e devem ser protegidos uns dos outros. Qualquer dos processos deve ser limitado quanto ao montante de memória que pode utilizar e às operações que pode executar em dispositivos como discos.
   c. Deve-se ter cuidado para evitar deadlocks entre processos no kernel e, portanto, os processos não fiquem esperando pelos recursos alocados uns dos outros.

2) [R] Descreva as diferenças entre o escalonamento de curto prazo, de médio prazo e de longo prazo.

   Resposta:

   a. **Curto prazo (scheduler da CPU)** – seleciona, a partir de jobs na memória, aqueles que estão prontos para serem executados, e aloca a CPU a eles.
   b. **Médio prazo** – utilizado especialmente com sistemas de tempo compartilhado como um nível de scheduling intermediário.
   c. **Longo prazo (scheduler de jobs)** – determina quais jobs são levados à memória para processamento.

   A principal diferença está na frequência da sua execução. O de curto prazo deve selecionar um novo processo com muita frequência. O de longo prazo é utilizado com muito menos frequência já que ele manipula jobs colocados no sistema e pode esperar que o job termine antes de admitir outro.

3) [R] Descreva as ações executadas por um kernel na mudança de contexto entre processos.

   Resposta:

   Em geral, o sistema operacional deve salvar o estado do processo em execução corrente e restaurar o estado do processo definido para entrar em execução em seguida. Salvar o estado de um processo normalmente inclui os valores de todos os registradores da CPU, além da alocação da memória.

4) [R] Explique por que programas são ditos serem entidades *passivas*, enquanto processos são entidades *ativas*.

   Resposta:

   Um programa é dito ser uma entidade passiva pois é simplesmente um arquivo contendo uma lista de instruções armazenadas. Tais instruções não tem qualquer “poder” a menos que o executável seja posto em execução pelo usuário ou pelo sistema. Quando um programa é executado, um processo é gerado. O processo é dito ser uma entidade ativa, pois, uma vez alocado a um processador, suas instruções poderão ser executadas. Além disso, um processo pode solicitar um recurso ao sistema operacional ou efetuar chamadas ao sistema.

5) [R] Cite e descreva os 5 estados básicos que um processo pode assumir num sistema operacional.

Resposta:

- **Novo (New):** O processo está sendo criado.
- **Executando (Running):** Instruções estão sendo executadas.
- **Esperando (Waiting):** O processo está esperando que ocorra algum evento (p.e. término de E/S).
- **Pronto (Ready):** O processo está esperando para ser atribuído a um processador.
- **Terminado (Terminated):** O processo terminou sua execução.

6) [R] Existem basicamente dois modelos de comunicação interprocesso (IPC). Quais são eles e como funcionam?

Resposta:

Memória Compartilhada: Processos se comunicam e "combinam" uma região de memória compartilhada. O formato e local dos dados são de responsabilidade completa dos processos envolvidos (SO não interfere). Processos são responsáveis por garantir que não estarão escrevendo no mesmo local simultaneamente. Mais rápida que a troca de mensagens por não exigir muitas chamadas ao sistema (baixa intervenção do kernel).

Troca de mensagens: Processos se comunicam entre si sem lançar mão de variáveis compartilhadas. Para isso eles estabelecem um link de comunicação (físico ou lógico) e chamam primitivas do tipo send() e receive() para enviar e receber dados.

7) [R] Defina passagem de mensagem com bloqueio/síncrona e sem bloqueio/assíncrona.

Resposta:

- **Bloqueio** é considerado **síncrono**
  - *Envio com bloqueio* deixa o emissor bloqueado até que a mensagem é recebida
  - *Recepção com bloqueio* deixa o receptor bloqueado até que a uma mensagem esteja disponível
- **Não bloqueio** é considerado **assíncrono**
  - *Envio sem bloqueio* faz com que o emissor envie a mensagem e continue
  - *Recepção sem bloqueio* faz com que o receptor receba uma mensagem válida ou nulo

Cap. 4: THREADS

1) [**] Forneça dois exemplos de programação em que a criação de vários threads proporcione melhor desempenho do que uma solução com um único thread.

Resposta:

(1) Um servidor Web que sirva cada solicitação em um thread separado. (2) Uma aplicação paralelizada, tal como uma multiplicação de matrizes, em que diferentes partes da matriz podem ser trabalhadas em paralelo. (3) Um programa de GUI interativo, tal como um depurador, em que são utilizados um thread para monitorar a entrada do usuário, outro thread para representar a aplicação em execução e um terceiro thread para monitorar o desempenho.

2) [**] Forneça dois exemplos de programação em que a criação de vários threads não proporcione melhor desempenho do que uma solução com um único thread.

Resposta:

(1) Qualquer tipo de programa sequencial não é um bom candidato a constituir threads. Um exemplo disso é um programa que calcula o retorno de uma taxa individual. (2) Outro exemplo é um programa “shell”, tal como o C-shell ou o shell Korn. Esse programa deve monitorar de perto seu próprio espaço de trabalho, tal como os arquivos abertos, as variáveis ambientais e o diretório de trabalho corrente.

3) [R] Cite duas diferenças entre os threads de nível de usuário e os de nível de kernel. Sob que circunstâncias um tipo é melhor do que o outro?

Resposta:

(1) Threads de nível de usuário são desconhecidos pelo kernel, enquanto o kernel está ciente dos threads de nível de kernel. (2) Em sistemas que utilizem mapeamento tanto M:1 quanto M:N, os threads de usuário são alocados ao schedule pela biblioteca de threads e o kernel organiza o schedule dos threads de nível de kernel. (3) Os threads do kernel não precisam estar associados a um processo, enquanto todo thread de usuário pertence a um processo. Os threads do kernel são geralmente mais dispendiosos para manter do que os threads de usuário, já que eles devem ser representados com uma estrutura de dados do kernel.

4) [R] Descreva as ações executadas por um kernel para mudar o contexto entre threads de nível de kernel.

Resposta:

A mudança de contexto entre threads do kernel normalmente requer o salvamento do valor dos registradores da CPU do thread expulso e a restauração dos registradores da CPU do novo thread alocado ao schedule.

5) Descreva as ações executadas por uma biblioteca de threads para mudar o contexto entre threads de nível de usuário.

Resposta:

A mudança de contexto entre threads do usuário é muito semelhante à mudança entre threads do kernel, embora ela dependa da biblioteca de threads e de como os threads de usuário são mapeados em threads do kernel. Em geral, a mudança de contexto entre threads de usuário envolve tirar um thread de usuário do seu LWP substituindo-o por outro thread. Essa ação normalmente envolve o salvamento e a restauração do estado dos registradores.

6) [R] Que recursos são usados quando um thread é criado? Em que eles diferem dos usados quando um processo é criado?

Resposta:

Como um thread é menor do que um processo, a criação do thread normalmente utiliza menos recursos do que a criação do processo. A criação de um processo requer a alocação de um bloco de controle de processo (PCB) que é uma estrutura de dados um tanto grande. O PCB inclui um mapa da memória, uma lista de arquivos abertos e variáveis ambientais. A alocação e o gerenciamento do mapa da memória são normalmente a atividade de maior consumo de tempo. A criação tanto de um thread de usuário quanto de um thread de kernel envolve a alocação de uma pequena estrutura de dados para manter o estado dos registradores, a pilha e as prioridades.

7) [**] Sob que circunstâncias uma solução com vários threads usando múltiplos threads de kernel fornece melhor desempenho do que uma solução com um único thread em um sistema com apenas um processador?

Resposta:

Quando um thread do kernel sofre um erro de página, outro thread do kernel pode ser permutado com ele para utilizar o tempo de intervalo de maneira útil. Um processo com um único thread, por outro lado, não será capaz de executar trabalho útil quando ocorrer um erro de página. Portanto, em cenários em que um programa pode sofrer de erros de página frequentes ou tenha que esperar por outros eventos do sistema, uma solução multithreads executaria melhor, mesmo em um sistema com um único processador.

8) [R] Qual dos componentes de estado de programa a seguir são compartilhados pelos threads em um processo com vários threads?

   a. Valores do registrador
   b. Memória do heap
   c. Variáveis globais
   d. Memória da pilha

Resposta:

Os threads de um processo multithreads compartilham a memória do heap e as variáveis globais. Cada thread tem seu conjunto separado de valores de registradores e uma pilha separada.

9) [**] Uma solução com vários threads usando múltiplos threads de nível de usuário pode obter melhor desempenho em um sistema multiprocessador do que em um sistema com um único processador? Explique.

Resposta:

Um sistema multithreads composto por múltiplos threads de nível de usuário não pode fazer uso dos diferentes processadores em um sistema multiprocessador, simultaneamente. O sistema operacional visualiza apenas um único processo e não organizará o schedule dos diferentes threads do processo em processadores separados. Consequentemente, não existe benefício de desempenho associado à execução de múltiplos threads de nível de usuário em um sistema multiprocessador.

10) [***] Considere um sistema multiprocessador e um programa com vários threads escrito com o uso do modelo muitos-para-muitos de criação de threads. Permita que a quantidade de threads de nível de usuário do programa seja maior do que a quantidade de processadores do sistema. Discuta as implicações de desempenho dos cenários a seguir.

a. A quantidade de threads do kernel alocada para o programa é menor do que a quantidade de processadores.
b. A quantidade de threads do kernel alocada para o programa é igual ao número de processadores.
c. A quantidade de threads do kernel alocada para o programa é maior do que a quantidade de processadores, mas menor do que a quantidade de threads de nível de usuário.

Resposta:

Quando o número de threads do kernel é menor do que o número de processadores, então alguns dos processadores poderiam permanecer ociosos, já que o escalonador mapeia, nos processadores, somente threads do kernel e não threads de nível de usuário. Quando o número de threads do kernel é exatamente igual ao número de processadores, então é possível que todos os processadores sejam utilizados simultaneamente. Entretanto, quando um thread do kernel é bloqueado dentro do kernel (devido a um erro de página ou ao invocar chamadas de sistema), o processador correspondente permanecerá ocioso. Quando existem mais threads do kernel do que processadores, um thread do kernel bloqueado pode ser expulso em favor de outro thread do kernel que esteja pronto para executar, aumentando assim a utilização do sistema multiprocessador.

Cap. 5: ESCALONAMENTO DE CPU

1) [R] Explique a diferença entre escalonamento com e sem preempção.

Resposta:

O escalonamento preemptivo permite que um processo seja interrompido no meio de sua execução, tomando a CPU e alocando-a a outro processo. O escalonamento não preemptivo garante que um processo deixe o controle da CPU somente quando terminar o seu pico de CPU corrente.

2) [*] Suponhamos que os processos a seguir chegassem para execução nos momentos indicados. Cada processo será executado durante o período de tempo listado. Ao responder às perguntas, use o escalonamento sem preempção e baseie todas as decisões nas informações disponíveis no momento em que a decisão tiver de ser tomada.

Processo	Tempo de Chegada	Duração do Pico
P_1	0,0	8
P_2	0,4	4
P_3	1,0	1

a. Qual é o tempo médio de turnaround desses processos com o algoritmo de escalonamento FCFS?

b. Qual é o tempo médio de turnaround desses processos com o algoritmo de escalonamento SJF?

c. O algoritmo SJF deveria melhorar o desempenho, mas observe que optamos por executar o processo P1 no momento 0 porque não sabíamos que dois processos mais curtos estavam para chegar. Calcule qual será o tempo médio de turnaround se a CPU for deixada ociosa durante a primeira unidade de tempo 1 para então o escalonamento SJF ser usado. Lembre que os processos P_1 e P_2 estão esperando durante esse tempo ocioso e, portanto, seu tempo de espera pode aumentar. Esse algoritmo poderia ser chamado de escalonamento de conhecimento futuro.

Resposta:

a. 10,53

b. 9,53

c. 6,86

Lembre-se de que o tempo de turnaround é igual à hora de término menos a hora de chegada, e, assim, você tem que subtrair as horas de chegada para calcular os tempos de turnaround. O FCFS será igual a 11 se você esquecer de subtrair a hora de chegada.

3) [*] Qual a vantagem de termos tamanhos diferentes para o quantum de tempo em níveis distintos de um sistema de enfileiramento em vários níveis?

Resposta:

Os processos que precisam de serviço mais frequente, como por exemplo os processos interativos, tais como os editores, podem estar em uma fila com um quantum de tempo pequeno. Os processos que não precisam de serviço frequente podem estar em uma fila com um quantum de tempo maior, requerendo menos mudanças de contexto para completar o processamento e fazendo, assim, uso mais eficiente do computador.

4) [**] Muitos algoritmos de escalonamento da CPU são parametrizados. Por exemplo, o algoritmo RR requer um parâmetro que indique a parcela de tempo. Filas com retroalimentação em vários níveis requerem parâmetros que definam a quantidade de filas, o algoritmo de escalonamento para cada fila, os critérios usados para mover processos entre as filas e assim por diante.

   Portanto, na verdade, esses algoritmos são conjuntos de algoritmos (por exemplo, o conjunto de algoritmos RR para todas as parcelas de tempo etc.). Um conjunto de algoritmos pode incluir outro (por exemplo, o algoritmo FCFS é o algoritmo RR com um quantum de tempo infinito).

   Que relação existe (se existir alguma) entre os conjuntos de pares de algoritmos a seguir?

   a. Por prioridades e SJF
   b. Filas com retroalimentação em vários níveis e FCFS
   c. Por prioridades e FCFS
   d. RR e SJF

   Resposta:

   a. O job mais curto tem a prioridade mais alta.
   b. O nível mais baixo do MLFQ (filas com retroalimentação em vários níveis) é o FCFS.
   c. O FCFS dá a prioridade mais alta ao job com o tempo de existência mais longo.
   d. Nenhuma

5) [*] Suponhamos que um algoritmo de escalonamento (do nível do escalonamento de CPU de curto prazo) favorecesse os processos que usaram o menor tempo do processador no passado recente. Por que esse algoritmo favorecerá programas limitados por I/O e ao mesmo tempo não deixará os programas limitados por CPU em estado permanente de inanição?

   Resposta:

Ele favorecerá os programas limitados por I/O por causa do pico de CPU relativamente curto requerido por eles; entretanto, os programas limitados por CPU não entrarão em inanição porque os programas limitados por I/O liberarão a CPU, relativamente com mais frequência, para fazer o seu I/O.

6) [*] Suponhamos que um sistema operacional mapeasse threads de nível de usuário para o kernel usando o modelo muitos-para-muitos e que o mapeamento fosse feito através do uso de LWPs. Além disso, o sistema permite que os desenvolvedores de programas criem threads de tempo real. É necessário vincular um thread de tempo real a um LWP?

   Resposta:

Sim, caso contrário um thread de usuário poderia ter que competir por um LWP disponível antes de ser realmente alocado ao escalonador. Vinculando o thread de usuário a um LWP, não existe latência durante a espera por um LWP disponível; o thread de usuário de tempo real pode ser imediatamente alocado ao escalonador.

7) [*] Por que é importante o escalonador distinguir programas limitados por I/O de programas limitados pela CPU?

   Resposta:

Programas limitados por I/O têm a propriedade de executarem apenas um pequeno volume de computação antes de executarem I/O. Tais programas normalmente não utilizam todo o seu quantum de CPU. Programas limitados pela CPU, por outro lado, utilizam todo o seu quantum sem executarem nenhuma operação de bloqueio de I/O. Consequentemente, se poderia fazer melhor uso dos recursos do computador

dando prioridade mais alta aos programas limitados por I/O e permitindo que eles executem na frente dos programas limitados pela CPU.

8) [**] Explique como os pares de critérios de escalonamento a seguir entram em conflito em certas configurações.
   a. Utilização da CPU e tempo de resposta
   b. Tempo médio de turnaround e tempo máximo de espera
   c. Utilização de dispositivos de I/O e utilização da CPU

   Resposta:

   a. **Utilização da CPU e tempo de resposta**: A utilização da CPU é aumentada se os overheads associados à mudança de contexto forem minimizados. Os overheads por mudança de contexto podem ser diminuídos com a execução menos frequente de mudanças de contexto. Isso pode, entretanto, resultar em aumento do tempo de resposta dos processos.
   b. **Tempo médio de turnaround e tempo máximo de espera**: O tempo médio de turnaround é minimizado pela execução das tarefas mais curtas primeiro. Tal política de escalonamento pode, entretanto, causar inanição das tarefas de execução mais demorada, aumentando assim seus tempos de espera.
   c. **Utilização de dispositivos de I/O e utilização da CPU**: A utilização da CPU é maximizada pela execução de tarefas limitadas pela CPU de execução demorada, sem executar mudanças de contexto. A utilização de dispositivos de I/O é maximizada pelo escalonamento de jobs limitados por I/O tão logo eles estejam prontos para executar, incorrendo assim nos overheads das mudanças de contexto.

9) [**] Considere o conjunto de processos a seguir, com a duração do pico de CPU dada em milissegundos:

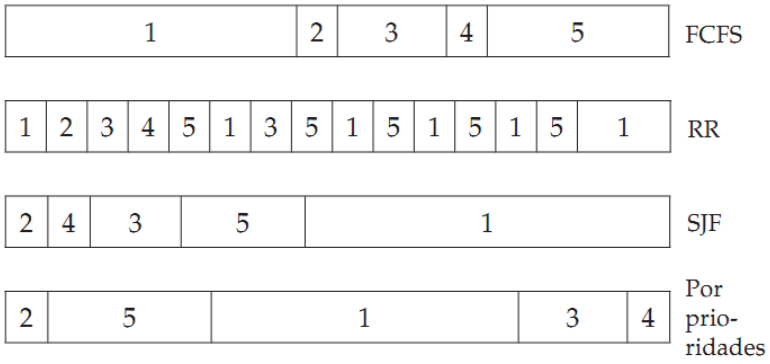
Processo	Duração do Pico	Prioridade
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

   Presume-se que os processos tenham chegado na ordem P_1, P_2, P_3, P_4, P_5, todos no momento 0.

   a. Desenhe quatro gráficos de Gantt que ilustrem a execução desses processos usando os algoritmos de escalonamento a seguir: FCFS, SJF, por prioridades sem preempção (um número de prioridade menor implica uma prioridade mais alta) e RR (quantum = 1).
   b. Qual é o tempo de turnaround de cada processo para cada um dos algoritmos de escalonamento da parte a?
   c. Qual é o tempo de espera de cada processo para cada um desses algoritmos de escalonamento?
   d. Qual dos algoritmos resulta no menor tempo médio de espera (para todos os processos)?

   Resposta:

   a. Os quatro gráficos de Gantt são:



b. Tempo de turnaround

	FCFS	RR	SJF	Por prioridades
P_1	10	19	19	16
P_2	11	2	1	1
P_3	13	7	4	18
P_4	14	4	2	19
P_5	19	14	9	6

c. Tempo de espera (tempo de turnaround menos tempo de pico)

	FCFS	RR	SJF	Por prioridades
P_1	0	9	9	6
P_2	10	1	0	0
P_3	11	5	2	16
P_4	13	3	1	18
P_5	14	9	4	1

d. Job mais curto primeiro

10) Qual dos algoritmos de escalonamento a seguir poderia resultar em inanição?

a. “Primeiro a entrar, primeiro a ser atendido”
b. Menor job primeiro
c. Round-Robin
d. Por prioridades

Resposta:

Os algoritmos de escalonamento job-mais-curto-primeiro e baseado-em-prioridades podem resultar em inanição.

11) [*] Considere uma variante do algoritmo de escalonamento RR em que as entradas da fila de prontos são ponteiros para os PCBs.

a. Qual seria o efeito da inserção de dois ponteiros para o mesmo processo na fila de prontos?
b. Cite duas grandes vantagens e duas desvantagens desse esquema.
c. Como você modificaria o algoritmo RR básico para obter o mesmo efeito sem os ponteiros duplicados?

Resposta:

a. O processo terá sua prioridade aumentada, já que, ao obter tempo com mais frequência, ele estará recebendo tratamento preferencial.

b. A vantagem é que os jobs de maior importância podem receber mais tempo ou, em outras palavras, tratamento com prioridade mais alta. A consequência, naturalmente, é que os jobs mais curtos seriam prejudicados.

c. Distribuindo um montante de tempo mais longo aos processos que mereçam prioridade mais alta. Em outras palavras, tendo dois ou mais quanta possíveis no esquema Round-Robin.

12) [**] Considere um sistema executando dez tarefas limitadas por I/O e uma tarefa limitada por CPU. Suponhamos que as tarefas limitadas por I/O emitissem uma operação de I/O a cada milissegundo de processamento da CPU e que cada operação de I/O levasse 10 milissegundos para ser concluída. Suponhamos também que o overhead da mudança de contexto fosse de 0,1 milissegundo e que todos os processos fossem tarefas de execução longa. Descreva a utilização da CPU para um escalonador round-robin quando:

a. O quantum de tempo é de 1 milissegundo

b. O quantum de tempo é de 10 milissegundos

Resposta:

a. O quantum de tempo é de 1 milissegundo: Independentemente de qual processo está para ser executado, o escalonador incorre no custo de mudança de contexto de 0,1 milissegundo para cada mudança de contexto. Isso resulta em uma utilização da CPU de 1 / 1,1 * 100 = 91%.

b. O quantum de tempo é de 10 milissegundos: As tarefas limitadas por I/O incorrem em uma mudança de contexto após utilizarem apenas 1 milissegundo do quantum de tempo. O tempo requerido para circular por todos os processos é, portanto, de 10 * 1,1 + 10,1 (já que cada tarefa limitada por I/O executa por 1 milissegundo e então incorre em mudança de contexto, enquanto a tarefa limitada pela CPU executa por 10 milissegundos antes de incorrer em mudança de contexto). A utilização da CPU é, portanto, de 20 / 21,1 * 100 = 94%.

13) [*] Explique as diferenças no grau de atuação dos algoritmos de escalonamento a seguir em favor de processos curtos:

a. FCFS

b. RR

c. Filas com retroalimentação em vários níveis

Resposta:

a. **FCFS** – atua contra os jobs curtos, já que quaisquer jobs curtos que cheguem após jobs longos terão um tempo de espera mais longo.

b. **RR** – trata todos os jobs igualmente (dando a eles picos de tempo de CPU iguais), e, assim, os jobs curtos serão capazes de deixar o sistema mais rápido, já que eles terminarão primeiro.

c. **Filas com retroalimentação em vários níveis** – funcionam de modo semelhante ao algoritmo RR; atuam favoravelmente em relação aos jobs curtos.

14) Usando o algoritmo de escalonamento do Windows XP, determine a prioridade numérica de cada um dos threads a seguir.

a. Um thread da classe REALTIME_PRIORITY_CLASS com uma prioridade relativa HIGHEST

b. Um thread da classe NORMAL_PRIORITY_CLASS com uma prioridade relativa NORMAL

c. Um thread da classe HIGH_PRIORITY_CLASS com uma prioridade relativa ABOVE_NORMAL

Resposta:

a. 26

b. 8

c. 14

Cap. 6: SINCRONISMO DE PROCESSOS

1) [R] Defina o conceito de Região Crítica (ou Seção Crítica) de um processo.

Resposta:

Um segmento de código que pode alterar dados (variáveis, tabelas, arquivos, etc...) comuns (ou seja, compartilhadas) a vários processos/threads é chamado de seção crítica (SC) ou região crítica (RC).

Só se pode garantir a consistência dos dados compartilhados se apenas um processo por vez adentrar a sua seção crítica. Ou seja, dois processos não podem executar suas seções críticas ao mesmo tempo.

2) As condições de corrida podem ocorrer em muitos sistemas de computação. Considere um sistema bancário com duas funções: `deposit(amount)` e `withdraw(amount)`. Essas duas funções recebem a quantia (amount) que deve ser depositada em uma conta bancária ou dela retirada. Suponha que exista uma conta bancária conjunta de um marido e sua esposa e, concorrentemente, o marido chamasse a função `withdraw()` e a esposa chamasse `deposit()`. Descreva como uma condição de corrida pode ocorrer e o que pode ser feito para impedir que ela ocorra.

Resposta:

Considere que o saldo da conta seja de $250,00 e o marido chame `withdraw(50)` e a esposa chame `deposit(100)`. Obviamente, o valor correto seria $300,00. Como essas duas transações serão serializadas, o valor local do saldo para o marido torna-se $200,00, mas antes que ele possa confirmar a transação, a operação `deposit(100)` tem lugar e atualiza o valor compartilhado do saldo para $300,00. Ao revertermos, então, ao marido, o valor do saldo compartilhado é posicionado como $200,00 – obviamente um valor incorreto.

3) [R] Quais os requisitos principais de qualquer solução para o problema da seção crítica?

Resposta:

- A apenas um processo é permitido estar dentro de sua seção crítica num dado instante (Exclusão Mútua).
- Nenhum processo que executa fora de sua região crítica pode bloquear outro processo (ex: processo para fora da sua seção crítica).
- Nenhuma suposição pode ser feita sobre as velocidades relativas dos processos ou sobre o número de CPUs no sistema.
- Nenhum processo pode ter que esperar eternamente para entrar em sua seção crítica ou lá ficar eternamente.

4) [R] O que é *busy wait*? Qual a consequência principal da solução que a possui?

Resposta:

Busy wait = espera ativa ou espera ocupada.

Se uma solução possui busy wait, basicamente o que ela faz é: Quando um processo quer entrar na sua R.C. ele verifica se a entrada é permitida. Se não for, ele espera em um laço (improdutivo) até que o acesso seja liberado. Ex: while (vez == OUTRO) do {nothing};

Consequência: desperdício de tempo de CPU.

5) [R] Explique como funciona o mecanismo de inibição/desativação de interrupções.

Resposta:

Usa um par de instruções do tipo DI (Disable Interrupt) / EI (Enable Interrupt). O processo desativa todas as interrupções imediatamente antes de entrar na sua RC, reativando-as imediatamente depois de sair dela. Com as interrupções desativadas, nenhum processo que está na sua RC pode ser interrompido, o que garante o acesso exclusivo aos dados compartilhados.

6) [*] Explique por que a implementação de primitivas de sincronização através da desativação de interrupções não é apropriada em um sistema com um único processador se os primitivas de sincronização tiverem de ser usados em programas de nível de usuário.

Resposta:

Se um programa de nível de usuário tem a capacidade de desabilitar interrupções, então ele poderá desabilitar a interrupção de timer e impedir que a comutação de contexto tenha lugar, permitindo assim que ele utilize o processador sem deixar que outros processos executem.

7) [R] Explique por que as interrupções não são apropriadas para a implementação de primitivas de sincronização em sistemas multiprocessadores.

Resposta:

Interrupções não são suficientes em sistemas multiprocessadores, já que a desabilitação de interrupções impede apenas que outros processos executem no processador no qual as interrupções foram desabilitadas; não há limites para a execução dos processos em outros processadores, e, portanto, o processo, ao desabilitar interrupções, não pode garantir acesso mutuamente exclusivo ao estado do programa.

8) [**] Os servidores podem ser projetados para limitar a quantidade de conexões abertas. Por exemplo, um servidor pode querer ter apenas N conexões de socket em um determinado momento. Assim que N conexões forem estabelecidas, ele não aceitará outra conexão até que uma conexão existente seja liberada. Explique como os semáforos podem ser usados por um servidor na limitação da quantidade de conexões concorrentes.

Resposta:

Um semáforo é inicializado com o número de conexões de socket abertas permitidas. Quando uma conexão é aceita, o método acquire() é chamado; quando uma conexão é liberada, o método release() é chamado. Se o sistema atinge o número de conexões de socket permitidas, chamadas subsequentes a acquire() serão bloqueadas até que uma conexão existente seja concluída e o método release() seja invocado.

9) [*] Demonstre que, se as operações de semáforo `wait()` e `signal()` não forem executadas atomicamente, a exclusão mútua pode ser violada.

Resposta:

Uma operação wait decrementa atomicamente o valor associado ao semáforo. Se duas operações wait forem executadas sobre um semáforo quando seu valor for igual a 1 e se as duas operações não forem executadas atomicamente, então é possível que as duas operações decrementem o valor do semáforo, violando assim a exclusão mútua.

10) Demonstre como podem ser implementadas as operações de semáforo wait() e signal() em ambientes multi-processadores com o uso da instrução TestAndSet(). A solução deve incorrer em um nível mínimo de espera em ação.

Resposta:

Eis o pseudocódigo para implementar as operações:

```
int guard = 0;
int semaphore_value = 0;
wait()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore value == 0) {
          // atomically add process to a queue of processes
          // waiting for the semaphore and set guard to 0;
    }else {
          semaphore_value--;
          guard = 0;
    }
}
signal()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore_value == 0 && there is a process on the wait queue)
          // wake up the first process in the queue of waiting processes
    else
          semaphore_value++;
    guard = 0;
}
```

11) Discuta a relação entre compartilhamento justo e throughput nas operações do problema dos leitores-escritores. Proponha um método para resolver o problema dos leitores-escritores sem causar inanição.

Resposta:

O throughput no problema dos leitores-escritores é aumentado pelo favorecimento de múltiplos leitores em contraposição à permissão de que um único escritor acesse exclusivamente os valores compartilhados. Por outro lado, o favorecimento dos leitores pode resultar em inanição para os escritores. A inanição no problema dos leitores-escritores pode ser evitada mantendo-se marcadores de tempo associados aos processos em espera. Quando um gravador concluir sua tarefa, ele despertará o processo que estiver esperando durante o maior período de tempo. Quando um leitor chegar e informar que outro leitor está acessando o banco de dados, ele somente entrará na seção crítica se não houver escritores em espera. Essas restrições garantiriam justiça.

12) [R] Cite e explique alguns problemas inerentes ao uso de semáforos na sincronização de processos.

Resposta:

- Troca da ordem em que são executadas as operações de `wait()` e `signal()` num mutex.

```
signal(mutex);
...
seção crítica
...
wait(mutex);
```

Nessa situação vários processos podem estar executando em suas seções críticas concorrentemente, violando o requisito de exclusão mútua. E, com vários processos rodando simultaneamente, pode ser muito difícil detectar esse erro.

- Substituição de `signal()` por `wait()`, ou seja:

```
wait(mutex);
```

```
...
seção crítica
...
wait(mutex);
```

Nesse caso, ocorrerá um deadlock.

- Omissão de um `wait()` ou `signal()`, ou ambos. Nesse caso, ou a exclusão mútua é violada ou ocorrerá um deadlock.

Cap. 7: DEADLOCKS

1) [*] Liste três exemplos de deadlocks que não estejam relacionados a um ambiente de sistema de computação.

   Resposta:

   - Dois carros atravessando uma ponte com uma única pista, vindo de direções opostas.
   - Uma pessoa descendo uma ladeira enquanto outra pessoa sobe a ladeira.
   - Dois trens viajando em direção um ao outro no mesmo trilho.
   - Dois carpinteiros que devem martelar pregos. Existem um único martelo e uma única caçamba de pregos. Ocorrerá um deadlock se um carpinteiro tiver o martelo e o outro carpinteiro, os pregos.

2) [**] Para pensar: Por que humanos não entram em deadlock?

   Resposta:

   **** Discussão em sala ****

3) [R] Quais os principais métodos gerais para tratamento de Deadlocks?

   Resposta:

   - Garantir que o sistema nunca entrará em um estado de deadlock.
     - Garantindo que pelo uma das condições necessárias não seja satisfeita.
   - Permitir que o sistema entre em um estado de deadlock e depois se recupere.
     - No entanto, o SO deverá ser capaz de determinar se aconteceu um deadlock.
   - Ignorar o problema e fingir que os deadlocks nunca ocorrem no sistema; usado pela maioria dos sistemas operacionais, incluindo UNIX e Windows.
     - Aparentemente drástico, mas se os deadlocks forem considerados raros (±1 vez/ano) é a opção menos dispendiosa.

4) Suponha que um sistema esteja em um estado inseguro. Demonstre ser possível que os processos concluam sua execução sem entrar em estado de deadlock.

   Resposta:

Um estado inseguro não necessariamente pode levar a um deadlock; ele apenas significa que não podemos garantir que o deadlock não ocorrerá. Assim, é possível que um sistema em estado inseguro possa ainda permitir que todos os processos terminem sem que ocorra um deadlock. Considere a situação em que um sistema tenha 12 recursos alocados entre os processos P_0, P_1 e P_2. Os recursos são alocados de acordo com a seguinte política:

	Max	Corrente	Necessidade
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

Correntemente, existem dois recursos disponíveis. Esse sistema está em estado inseguro, já que o processo P1 poderia se completar, liberando um total de quatro recursos. Mas não podemos garantir que os

processos P0 e P2 possam ser concluídos. Entretanto, é possível que um processo libere recursos antes de solicitar mais. Por exemplo, o processo P2 poderia liberar um recurso, aumentando assim o número total de recursos para cinco. Isso permitiria a conclusão do processo P0, que liberaria um total de nove recursos para permitir que o processo P2 também se complete.

5) [**] Um sistema pode detectar que algum de seus processos está sofrendo de *starvation* (inanição)? Se você responder "sim", explique como ele pode fazer isso. Se responder "não", explique como o sistema pode lidar com o problema da inanição.

Resposta:

A inanição é um tópico difícil de definir, já que pode significar coisas diferentes para sistemas diferentes. Para fins desta questão, definiremos inanição como a situação em que um processo deve esperar além de um período de tempo razoável — talvez indefinidamente — antes de receber um recurso solicitado. Um modo de detectar inanição seria primeiro identificar um período de tempo — T — que seja considerado razoável. Quando um processo solicitar um recurso, será iniciado um timer. Se o tempo transcorrido exceder T, então o processo será considerado em inanição.

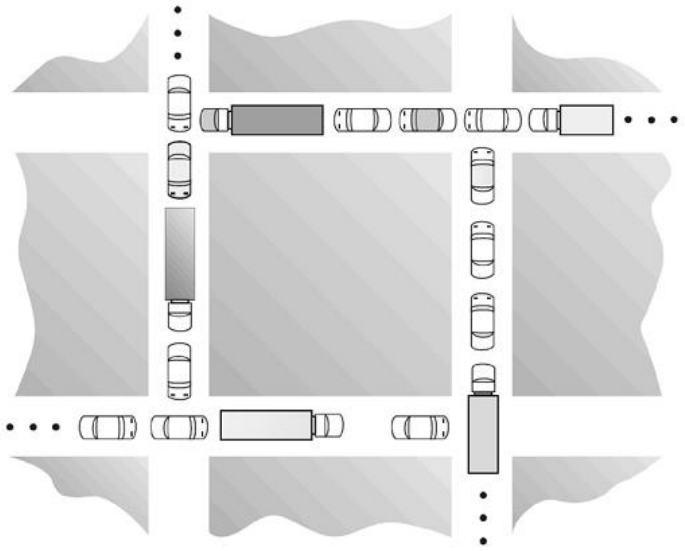
Uma estratégia para lidar com a inanição seria adotar uma política em que os recursos fossem atribuídos somente ao processo que esteve esperando por mais tempo. Por exemplo, se o processo Pa esteve esperando pelo recurso X por mais tempo do que o processo Pb, a solicitação do processo Pb seria adiada até que a solicitação do processo Pa tenha sido atendida. Outra estratégia seria menos estrita do que a que acabou de ser mencionada. Nesse cenário, um recurso pode ser concedido a um processo que esperou menos do que outro processo, cuidando para que o outro processo não entre em inanição. Entretanto, se o outro processo for considerado em inanição, sua solicitação será atendida primeiro.

6) [*] É possível ocorrer um deadlock envolvendo apenas um processo (com uma única thread, naturalmente)? Explique sua resposta.

Resposta:

Não. Isso deriva diretamente da condição de espera em um ciclo fechado (o deadlock depende das características de dois ou mais programas diferentes e dos respectivos processos, em execução ao mesmo tempo).

7) Considere o deadlock de tráfego mostrado na figura abaixo.
   a. Demonstre que as quatro condições necessárias para a ocorrência do deadlock estão presentes nesse exemplo.
   b. Defina uma regra simples para impedir deadlocks nesse sistema



Resposta:

a. As quatro condições necessárias para um deadlock são (1) exclusão mútua; (2) manutenção e espera; (3) não preempção e (4) espera circular. A condição de exclusão mútua se justifica já que apenas um carro pode ocupar espaço em uma rodovia. A condição de manutenção e espera ocorre quando um carro mantém seu lugar na rodovia enquanto espera para avançar. Um carro não pode ser removido (isto é, sofrer preempção) de sua posição na rodovia. Finalmente, existe realmente uma espera circular enquanto cada carro está esperando que um carro subsequente avance. A condição de espera circular também é facilmente observável no gráfico.

b. Uma regra simples que poderia evitar esse deadlock de tráfego é que um carro não pode avançar em um cruzamento se estiver claro que ele não será capaz de desocupar o cruzamento de imediato.

8) [*] Considere a situação de deadlock que pode ocorrer no problema dos filósofos glutões quando os filósofos obtêm os pausinhos um de cada vez. Discuta como as quatro condições necessárias para a ocorrência do deadlock estão presentes nesta situação. Discuta como os deadlocks poderiam ser evitados com a eliminação de qualquer uma das quatro condições necessárias.

Resposta:

O deadlock é possível porque as quatro condições necessárias estão presentes da seguinte maneira: 1) a exclusão mútua é requerida para os pausinhos, 2) os filósofos mantêm o pausinho em mãos enquanto esperam pelo outro pausinho, 3) não existe preempção de pausinhos no sentido de que um pausinho alocado a um filósofo não pode ser tirado à força e 4) existe a possibilidade de espera circular. Os deadlocks poderiam ser evitados superando-se as condições da seguinte maneira: 1) permitindo o compartilhamento simultâneo de pausinhos, 2) fazendo os filósofos desistirem do primeiro pausinho se não conseguirem obter o outro pausinho, 3) permitindo que os pausinhos sejam tirados à força se um filósofo retiver um pausinho por um longo período de tempo e 4) obrigando a numeração dos pausinhos, obtendo sempre o pausinho de número mais baixo antes de obter o de número mais alto.

9) [***] Considere um sistema composto por quatro recursos do mesmo tipo que são compartilhados por três processos, cada um deles precisando de no máximo dois recursos. Demonstre que o sistema está livre de deadlocks.

Resposta:

Suponha que o sistema esteja em deadlock. Isso implica que cada processo está mantendo um recurso e está esperando por mais um. Como existem três processos e quatro recursos, um processo deve ser capaz de

obter dois recursos. Esse processo não requererá mais recursos e, portanto, ele retornará seus recursos quando utilizados.

10) [***] Considere a versão do problema dos filósofos glutões em que os pausinhos são colocados no centro da mesa e qualquer par deles pode ser usado por um filósofo. Assuma que as solicitações de pausinhos são feitas uma de cada vez. Descreva uma regra simples para determinar se uma solicitação específica pode ser atendida sem causar deadlock dada a presente alocação de pausinhos para os filósofos.

Resposta:

A seguinte regra evita o deadlock: quando um filósofo fizer uma solicitação pelo primeiro pausinho, não atenda a solicitação se **não existir outro filósofo com dois pausinhos** (neste caso, garante que pelo menos um filósofo está comendo e, portanto, devolverá os pausinhos ao terminar) e **se existir apenas um pausinho sobrando** (indicando que já existe pelo menos um filósofo esperando por um pausinho para completar o par).

11) [**] Considere novamente a situação do exercício anterior. Assuma agora que cada filósofo precise de três pausinhos para comer. As solicitações de recursos ainda são emitidas uma de cada vez. Descreva algumas regras simples para determinar se uma solicitação específica pode ser atendida sem causar um deadlock dada esta alocação de chopsctiks para os filósofos.

Resposta:

Quando um filósofo solicitar um pausinho, atenda a solicitação se: 1) o filósofo tiver dois pausinhos e existir pelo menos um pausinho sobrando, 2) o filósofo tiver um pausinho e existem pelo menos dois pausinhos sobrando, 3) existir pelo menos um pausinho sobrando e existir pelo menos um filósofo com três pausinhos, 4) o filósofo não tiver pausinhos, existirem dois pausinhos sobrando e existir pelo menos um outro filósofo com dois pausinhos.

Cap. 8: MEMÓRIA PRINCIPAL

1) [R] Cite duas diferenças entre endereços lógicos e físicos.

Resposta:

Um endereço lógico não se refere a um endereço real existente; ele se refere a um endereço abstrato num espaço de endereçamento abstrato. Compare isso com um endereço físico que se refira a um endereço físico real em memória. Um endereço lógico é gerado pela CPU e é traduzido para um endereço físico pela unidade de gerenciamento da memória (MMU). Portanto, endereços físicos são gerados pela MMU.

2) [*] Considere um sistema em que um programa possa ser separado em duas partes: código e dados. A CPU sabe se deseja uma instrução (busca de instrução) ou dados (busca ou armazenamento de dados). Portanto, dois pares de registradores base-limite são fornecidos: um para instruções e outro para dados. O par de registradores base-limite de instruções é automaticamente somente de leitura; logo, os programas podem ser compartilhados entre diferentes usuários. Discuta as vantagens e desvantagens desse esquema.

Resposta:

A vantagem principal desse esquema é que ele é um mecanismo efetivo para compartilhamento de código e dados. Por exemplo, apenas uma cópia de um editor ou de um compilador precisa ser mantida em memória, e esse código pode ser compartilhado por todos que necessitem acessar o código do editor ou do compilador. Outra vantagem é a proteção do código contra modificações erradas. A única desvantagem é que o código e os dados devem ser separados, o que usualmente é anexado a um código gerado pelo compilador.

3) [*] Por que os tamanhos de página são sempre potências de 2?

Resposta:

Lembre-se de que a paginação é implementada dividindo-se um endereço em uma página e um número de deslocamento. É mais eficiente quebrar o endereço em X bits de página e Y bits de deslocamento, em vez de executar aritmética sobre o endereço para calcular o número da página e o deslocamento. Como cada posição de bit representa uma potência de 2, a divisão do endereço em bits resulta em um tamanho de página que é uma potência de 2.

4) [*] Considere um espaço de endereçamento lógico de 64 páginas com 1.024 palavras cada, mapeado para uma memória física de 32 quadros.

   a. Quantos bits há no endereço lógico?
   b. Quantos bits há no endereço físico?

   Resposta:

   a. Endereço lógico: 16 bits (2^6 x 2^{10})
   b. Endereço físico: 15 bits (2^5 x 2^{10})

5) [*] Qual é o efeito de permitir que duas entradas em uma tabela de páginas apontem para o mesmo quadro de página na memória? Explique como esse efeito poderia ser usado para diminuir o período de tempo necessário para copiar uma grande quantidade de memória de um local para outro. Que efeito a atualização de algum byte em uma página teria na outra página?

Resposta:

Ao permitir que duas entradas em uma tabela de páginas apontem para o mesmo quadro de página em memória, os usuários poderão compartilhar código e dados. Se o código for reentrante, muito espaço de memória poderá ser salvo por meio do uso compartilhado de programas grandes, tais como editores de texto, compiladores e sistemas de bancos de dados. A "cópia" de grandes montantes de memória poderia ser efetivada tendo diferentes tabelas de páginas apontando para a mesma localização de memória. Entretanto, o compartilhamento de código reentrante ou de dados significa que qualquer usuário que tenha acesso ao código poderá modificá-lo, e essas modificações poderiam se refletir na "cópia" do outro usuário.

6) [*] Descreva um mecanismo pelo qual um segmento poderia pertencer ao espaço de endereçamento de dois processos diferentes.

Resposta:

Como as tabelas de segmentos são uma coleção de registradores base-limite, os segmentos podem ser compartilhados quando as entradas na tabela de segmentos de dois jobs diferentes apontam para a mesma localização física. As duas tabelas de segmentos devem ter ponteiros base idênticos, e o número do segmento compartilhado deve ser o mesmo nos dois processos.

7) [R] Explique a diferença entre fragmentação interna e externa.

Resposta:

Fragmentação interna é a área em uma região ou uma página que não é utilizada pelo job que ocupa aquela região ou página. Esse espaço fica indisponível para uso até que o job termine e a página ou a região seja liberada.

8) Considere o processo a seguir para a geração de binários. Um compilador é utilizado para gerar o código-objeto para módulos individuais, e um link editor é usado para combinar vários módulos-objeto em um único binário de programa. Como o link editor altera a vinculação de instruções e dados a endereços de memória? Que informações precisam ser passadas do compilador para o link editor para facilitar as tarefas de vinculação da memória do link editor?

Resposta:

O link editor precisa substituir endereços simbólicos não resolvidos pelos endereços reais associados às variáveis, no binário final do programa. Para fazer isso, os módulos devem rastrear as instruções que se referem a símbolos não resolvidos. Durante a vinculação, é atribuída a cada módulo uma sequência de endereços no binário global do programa, e, quando isso é executado, as referências a símbolos não resolvidas exportadas por esse binário podem ser remendadas em outros módulos, já que todos os outros módulos conteriam a lista de instruções que precisam ser remendadas.

9) [*] Dadas cinco partições de memória de 100 KB, 500 KB, 200 KB, 300 KB e 600 KB (em ordem), como os algoritmos *first-fit*, *best-fit* e *worst-fit* alocariam processos de 212 KB, 417 KB, 112 KB e 426 KB (em ordem)? Que algoritmo faz o uso mais eficiente da memória?

Resposta:

first-fit:

212 K são alocados na partição de 500 K

417 K são alocados na partição de 600 K

112 K são alocados na partição de 288 K (nova partição de 288 K = 500 K – 212 K)

426 K devem esperar

best-fit:

212 K são alocados na partição de 300 K

417 K são alocados na partição de 500 K

112 K são alocados na partição de 200 K

426 K são alocados na partição de 600 K

worst-fit:

212 K são alocados na partição de 600 K

417 K são alocados na partição de 500 K

112 K são alocados na partição de 388 K

426 K devem esperar

Neste exemplo, o *best-fit* leva à melhor alocação

10) [**] A maioria dos sistemas permite que um programa aloque mais memória a seu espaço de endereçamento durante a execução. A alocação de dados nos segmentos de programa do heap é um exemplo desse tipo de alocação de memória. O que é necessário para suportar a alocação de memória dinâmica nos seguintes esquemas?
    a. Alocação de memória contígua
    b. Segmentação pura
    c. Paginação pura

    Resposta:

    a. **Alocação de memória contígua**: pode requerer a realocação do programa inteiro, já que não existe espaço suficiente para que o espaço alocado de memória do programa cresça.
    b. **Segmentação pura**: também pode requerer realocação do segmento que precisa ser estendido, já que não existe espaço suficiente para que o espaço alocado de memória do segmento cresça.
    c. **Paginação pura**: é possível a alocação incremental de novas páginas nesse esquema sem requerer realocação do espaço de endereçamento do programa

11) [**] Compare os esquemas de alocação de memória contígua, de segmentação pura e de paginação pura para a organização da memória em relação às questões a seguir:
    a. Fragmentação externa
    b. Fragmentação interna
    c. Possibilidade de compartilhar código entre processos

    Resposta:

O esquema de alocação de memória contígua sofre de fragmentação externa, já que os espaços de endereços são alocados contiguamente e geram intervalos à medida que processos antigos morrem e novos processos são iniciados. Ele também não permite que os processos compartilhem código, pois um segmento de memória virtual do processo não é quebrado em segmentos granulares não contíguos. A segmentação pura também sofre de fragmentação externa, já que um segmento de um processo é acomodado contiguamente na memória física e ocorre a fragmentação à medida que segmentos de processos mortos são substituídos por segmentos de novos processos. A segmentação, entretanto, habilita os processos a compartilharem código; por exemplo, dois processos diferentes podem compartilhar um segmento de código, mas com segmentos de dados distintos. A paginação pura não sofre de segmentação externa, mas sofre de fragmentação interna. Os processos são alocados em granularidade de páginas, e, se uma página não for completamente utilizada, isso resultará em fragmentação interna e no desperdício do espaço correspondente. A paginação também habilita os processos a compartilharem código devido à granularidade das páginas.

12) [***] Em um sistema com paginação, um processo não pode acessar memória que ele não possui. Por quê? Como o sistema operacional poderia permitir o acesso a outra memória? Por que ele deveria ou não deveria fazer isso?

Resposta:

Um endereço em um sistema de paginação é um número lógico de página e um deslocamento. A página física é encontrada buscando-se em uma tabela com base no número lógico da página para gerar um número físico da página. Como o sistema operacional controla o conteúdo dessa tabela, ele pode limitar um processo a acessar apenas as páginas físicas alocadas a ele. Não há maneira de um processo se referir a uma página que ele não possua porque a página não estará na tabela de páginas. Para permitir tal acesso, um sistema operacional simplesmente precisa permitir que entradas para a memória que não é do processo sejam adicionadas à tabela de páginas do processo. Isso é útil quando dois ou mais processos precisam trocar dados – eles apenas leem e gravam nos mesmos endereços físicos (que podem estar em endereços lógicos variados). Isso favorece uma comunicação interprocessos muito eficiente.

13) [**] Compare a paginação com a segmentação no que diz respeito à quantidade de memória requerida pelas estruturas de tradução de endereços para converter endereços virtuais em endereços físicos.

Resposta:

A paginação requer mais overhead de memória para manter as estruturas de tradução. A segmentação requer apenas dois registradores por segmento: um para manter a base do segmento e outro para manter o tamanho do segmento. A paginação, por outro lado, requer uma entrada por página e essa entrada fornece o endereço físico no qual a página está localizada.

14) [*] Os programas binários de muitos sistemas são, tipicamente, estruturados como descrito a seguir. O código é armazenado começando com um pequeno endereço virtual fixo, como 0. O segmento de código é seguido pelo segmento de dados que é usado para armazenar as variáveis do programa. Quando o programa começa a ser executado, a pilha é alocada na outra extremidade do espaço de endereços virtuais e pode crescer em direção a endereços virtuais menores. Qual é a importância dessa estrutura para os esquemas a seguir?

a. Alocação de memória contígua
b. Segmentação pura
c. Paginação pura

Resposta:

a. A alocação de memória contígua requer que o sistema operacional aloque todo o espaço de endereçamento virtual ao programa quando ele inicia sua execução. Esse espaço pode ser muito maior do que os requisitos de memória reais do processo.
b. A segmentação pura dá ao sistema operacional flexibilidade para atribuir uma pequena extensão a cada segmento em tempo de inicialização do programa e estender o segmento se necessário.
c. A paginação pura não requer que o sistema operacional aloque a extensão máxima do espaço de endereçamento virtual a um processo em tempo de inicialização, mas ainda exige que o sistema operacional aloque uma grande tabela de páginas abrangendo todo o espaço de endereçamento virtual do programa. Quando um programa precisa estender a pilha ou o heap, precisa alocar uma nova página, mas a entrada na tabela de páginas correspondente é pré-alocada.

15) [*] Supondo um tamanho de página de 1 KB, quais são os números de página e deslocamentos para as referências de endereço a seguir (fornecidas como números decimais):

a. 2375

b. 19366
c. 30000
d. 256
e. 16385

Resposta:

a. página = 2; deslocamento = 327
b. página = 18; deslocamento = 934
c. página = 29; deslocamento = 304
d. página = 0; deslocamento = 256
e. página = 14; deslocamento = 1

16) [**] Considere um sistema de computação com um endereço lógico de 32 bits e um tamanho de página de 4 KB. O sistema suporta até 512 MB de memória física. Quantas entradas existem em cada uma das tabelas de páginas a seguir?

a. Uma tabela de páginas convencional com um único nível
b. Uma tabela de páginas invertida

Resposta:

a. $2^{32} / 2^{12} = 2^{20} = 1.048.576$ entradas
b. $2^{29} / 2^{12} = 2^{17} = 131.072$ entradas

17) [*] Considere um sistema de paginação com a tabela de páginas armazenada em memória.

a. Se uma referência à memória levar 200 nanossegundos, quanto tempo levará uma referência à memória paginada?
b. Se adicionarmos TLBs e 75 por cento de todas as referências à tabela de páginas estiverem nos TLBs, qual é o tempo efetivo de referência à memória? (Suponha que a busca de uma entrada na tabela de páginas nos TLBs leve um período de tempo igual a 0, se a entrada estiver aí.)

Resposta:

a. 400 nanossegundos: 200 nanossegundos para acessar a tabela de páginas e 200 nanossegundos para acessar a palavra na memória.
b. Tempo de acesso efetivo = 0,75 x (200 nanossegundos) + 0,25 x (400 nanossegundos) = 250 nanossegundos

18) [**] Explique por que é mais fácil compartilhar um módulo reentrante quando é usada a segmentação em vez da paginação pura.

Resposta:

Como a segmentação é baseada em uma divisão lógica da memória em vez de física, segmentos de qualquer tamanho podem ser compartilhados com apenas uma entrada nas tabelas de segmentos de cada usuário. Com a paginação, deve existir uma entrada comum nas tabelas de páginas para cada página que é compartilhada.

19) [*] Considere a tabela de segmentos a seguir:

Segmento	Base	Tamanho
0	219	600
1	2300	14

2		
90	100	
3	1327	580
4	1952	96

Quais são os endereços físicos para os seguintes endereços lógicos?

a. <0, 430>
b. <1, 10>
c. <2, 500>
d. <3, 400>
e. <4, 112>

Resposta:

a. 219 + 430 = 649
b. 2300 + 10 = 2310
c. referência ilegal, exceção para o sistema operacional
d. 1327 + 400 = 1727
e. referência ilegal, exceção para o sistema operacional

20) [R] Qual é a finalidade da paginação das tabelas de páginas?

Resposta:

Em certas situações, as tabelas de páginas podem se tornar tão grandes que a paginação das tabelas de páginas pode simplificar o problema de alocação da memória (garantindo que tudo seja alocado como páginas de tamanho fixo em oposição às porções de tamanho variável) e também habilitar a permuta de porções da tabela de páginas que não são correntemente utilizadas.

Cap. 9: MEMÓRIA VIRTUAL

1) [R] Sob que circunstâncias ocorrem erros de página? Descreva as ações executadas pelo sistema operacional quando ocorre um erro de página.

Resposta:

Um erro de página ocorre quando tem lugar um acesso a uma página que não foi levada à memória. O sistema operacional verifica o acesso à memória abortando o programa se o acesso for inválido. Se o acesso for válido, é alocado um quadro livre e solicitado I/O para ler a página necessária no quadro livre. Após o término do I/O, a tabela de processos e a tabela de páginas são atualizadas e a instrução é reiniciada.

2) [**] Suponha que você tenha uma sequência de referências de página para um processo com ***m*** quadros (todos inicialmente vazios). A sequência de referências de página tem tamanho ***p***; ***n*** números de páginas distintas ocorrem nela. Responda a estas perguntas para qualquer algoritmo de substituição de páginas:

   a. O que é um limite inferior para o número de erros de página?
   b. O que é um limite superior para o número de erros de página?

Resposta:

   a. n (Se as páginas são distintas, todas falharão)
   b. p (É o pior caso, em que mesmo as páginas anteriormente lidas são removidas da memória. Todas as referências falham)

3) [**] Quais das técnicas e estruturas de programação a seguir são “boas” para um ambiente paginado por demanda? Quais delas são “ruins”? Explique suas respostas.

   a. Pilha
   b. Tabela de símbolos com hash
   c. Busca sequencial
   d. Busca binária
   e. Código puro
   f. Operações em vetores
   g. Acesso indireto

Resposta:

   a. Pilha — boa
   b. Tabela de símbolos com hash — ruim
   c. Busca sequencial — boa
   d. Busca binária — ruim
   e. Código puro — boa
   f. Operações em vetores — boa
   g. Acesso indireto — ruim

4) [R] Considere os algoritmos de substituição de páginas a seguir. Classifique estes algoritmos do melhor para o pior no que diz respeito a sua taxa de erros de página. Separe os algoritmos afetados pela anomalia de Belady daqueles que não o são.

a. Substituição LRU
b. Substituição FIFO
c. Substituição ótima
d. Substituição da segunda chance

Resposta:

Rank	Algoritmo	Sofre da anomalia de Belady
1	Ótimo	não
2	LRU	não
3	Segunda chance	sim
4	FIFO	sim

5) [*] Quando a memória virtual é implementada em um sistema de computação, há certos custos associados à técnica e certos benefícios. Liste os custos e os benefícios. É possível que os custos superem os benefícios? Se for, que medidas podem ser tomadas para garantir que isso não ocorra?

Resposta:

Os custos são hardware adicional e tempo de acesso mais lento. Os benefícios são boa utilização da memória e espaço de endereçamento lógico maior do que o espaço de endereçamento físico.

6) [***] Considere o array bidimensional A em C/C++:

```
int A[100] [100]
```

onde `A[0][0]` está na localização 200 em um sistema de memória paginada com páginas de tamanho 200. Um pequeno processo que manipula a matriz reside na página 0 (localizações 0 a 199). Assim, toda busca de instrução ocorrerá a partir da página 0.
Para três quadros de páginas, quantos erros de página serão gerados pelos seguintes loops de inicialização do array, utilizando a substituição LRU e supondo que o quadro de páginas 1 contenha o processo e os outros dois estejam inicialmente vazios?

a)
```
for (int i = 0; i < 100; i++)
      for (int j = 0; j < 100; j++)
            A[i] [j] = 0;
```

b)
```
for (int j = 0; j < 100; j++)
      for (int i = 0; i < 100; i++)
            A[i] [j] = 0;
```

Dica: Se não já não souber, pesquise a forma como é feita a alocação de matrizes bidimensionais em C. Ou seja, como é feito o mapeamento 2D → 1D (a memória é um como um array unidimensional). Você verá que é uma excelente lição de programação a ser aprendida!

Resposta:

O preenchimento se dá alocando duas linhas por quadro. Logo:

a. 50 (Preenchimento em linhas - É gerada uma falha de página a cada duas linhas, apenas)
b. 5.000 (Preenchimento estar se dando em colunas – Mesmas falhas do caso ‘a’ porém ocorrendo para todas as 100 colunas, logo 50 x 100 = 5000)

7) [R] Considere a seguinte sequência de referências de página:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Quantos erros de página ocorreriam para os algoritmos de substituição a seguir, considerando um, dois, três, quatro, cinco, seis e sete quadros? Lembre-se de que todos os quadros estão inicialmente vazios, de modo que a primeira página de cada um implicará em um erro de página.

- Substituição LRU
- Substituição FIFO
- Substituição ótima

Resposta:

Número de quadros	LRU	FIFO	Ótimo
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

8) [*] Você imaginou um novo algoritmo de substituição de páginas que acha que pode ser ótimo. Em alguns casos de teste distorcidos, ocorre a anomalia de Belady. O novo algoritmo é ótimo? Explique sua resposta.

Resposta:

Não. Um algoritmo ótimo não sofreria da anomalia de Belady porque — por definição — um algoritmo ótimo substitui a página que não será utilizada por mais tempo. A anomalia de Belady ocorre quando um algoritmo de substituição de páginas expulsa uma página que será necessária em futuro imediato. Um algoritmo ótimo não teria selecionado essa página.

9) [*] A segmentação é semelhante à paginação, mas utiliza “páginas” de tamanho variável. Defina dois algoritmos de substituição de segmentos baseados nos esquemas de substituição de páginas FIFO e LRU. Lembre-se de que, como os segmentos não são do mesmo tamanho, o segmento escolhido para ser substituído pode não ser grande o bastante para deixar localizações consecutivas suficientes para o segmento requerido. Considere estratégias para sistemas em que os segmentos não possam ser relocados e estratégias para sistemas onde isso possa ocorrer.

Resposta:

FIFO. Encontre o primeiro segmento grande o suficiente para acomodar o segmento que entra. Se a relocação não for possível e nenhum segmento for suficientemente grande, selecione uma combinação de segmentos cujas memórias sejam contíguas, que sejam “os mais próximos ao primeiro da lista” e que possam acomodar o novo segmento. Se a relocação for possível, reorganize a memória de modo que os primeiros N segmentos suficientemente grandes para o segmento que entra fiquem contíguos na memória. Adicione qualquer espaço restante à lista de espaços livres em ambos os casos.

LRU. Selecione o segmento que não tenha sido utilizado pelo período de tempo mais longo e que seja grande o suficiente, adicionando qualquer espaço restante à lista de espaços livres. Se nenhum segmento for suficientemente grande, selecione uma combinação dos segmentos “mais antigos” que estejam contíguos na memória (se a relocação não estiver disponível) e que sejam suficientemente grandes. Se a relocação estiver

disponível, reorganize os N segmentos mais antigos para que fiquem contíguos na memória e substitua-os pelo novo segmento.

10) [**] Considere um sistema de computação paginado por demanda em que o grau de multiprogramação esteja correntemente fixado em quatro. O sistema foi recentemente medido para determinar a utilização da CPU e o disco de paginação. Os resultados são uma das alternativas apresentadas a seguir. O que está acontecendo em cada caso? É possível aumentar o grau de multiprogramação para aumentar a utilização da CPU? A paginação está ajudando?
    a. Utilização da CPU, 13 por cento; utilização do disco, 97 por cento
    b. Utilização da CPU, 87 por cento; utilização do disco, 3 por cento
    c. Utilização da CPU, 13 por cento; utilização do disco, 3 por cento

    Resposta:

    a. Está ocorrendo atividade improdutiva (*Thrashing*). Deve-se diminuir o grau de multiprogramação.
    b. A utilização da CPU está suficientemente alta para deixar as coisas como estão.
    c. Aumentar o grau de multiprogramação.

11) [**] Suponha que um programa tivesse acabado de referenciar um endereço na memória virtual. Descreva um cenário em que cada uma das situações abaixo possa ocorrer. (Se tal cenário não puder ocorrer, explique por quê.)
    - Erro de TLB sem erro de página
    - Erro de TLB e erro de página
    - Sucesso do TLB sem erro de página
    - Sucesso do TLB e erro de página

    Resposta:

    - **Erro de TLB sem erro de página**: a página foi trazida para a memória, mas foi removida do TLB
    - **Erro de TLB e erro de página**: ocorreu um erro de página
    - **Sucesso do TLB sem erro de página**: a página está na memória e no TLB. A maioria seria, provavelmente, uma referência recente.
    - **Sucesso do TLB e erro de página**: não pode ocorrer. O TLB é um cache da tabela de páginas. Se uma entrada não está na tabela de páginas, não estará no TLB.

12) [*] Considere um sistema que utilize paginação por demanda pura.
    a. Quando um processo inicia sua execução pela primeira vez, como você caracterizaria a taxa de erros de página?
    b. Uma vez que o conjunto ativo para um processo seja carregado em memória, como você caracterizaria a taxa de erros de página?
    c. Suponha que um processo altere sua localidade e o tamanho do novo conjunto ativo seja grande demais para ser armazenado na memória livre disponível. Identifique algumas opções que os projetistas de sistemas poderiam escolher para manipular essa situação.

    Resposta:

    a. Inicialmente, bastante alta, já que as páginas necessárias ainda não foram carregadas na memória.
    b. Pode ser bem baixa, já que todas as páginas necessárias estão carregadas na memória.
    c. (1) Ignorá-la; (2) obter mais memória física; (3) reclamar páginas mais agressivamente devido à alta taxa de erros de página.

13) [**] Um determinado computador fornece a seus usuários um espaço de memória virtual de 2^{32} bytes. O computador tem 2^{18} bytes de memória física. A memória virtual é implementada pela paginação e o tamanho da página é de 4.096 bytes. Um processo de usuário gera o endereço virtual 11.123.456. Explique como o sistema estabelece a localização física correspondente. Faça a distinção entre operações de software e operações de hardware.

Resposta:

O endereço virtual em forma binária é

0001 0001 0001 0010 0011 0100 0101 0110

Como o tamanho da página é 2^{12}, o tamanho da tabela de páginas é 2^{20}. Portanto, os 12 bits de baixa ordem "0100 0101 0110" são utilizados como o deslocamento dentro da página, enquanto os 20 bits remanescentes "0001 0001 0001 0010 0011" são utilizados como o deslocamento na tabela de páginas.

14) Suponha que tenhamos uma memória paginada por demanda. A tabela de páginas é mantida em registradores. São necessários 8 milissegundos para manipular um erro de página se um quadro vazio estiver disponível ou se a página substituída não foi modificada e 20 milissegundos se a página substituída foi modificada. O tempo de acesso à memória é de 100 nanossegundos. Suponha que a página a ser substituída seja modificada 70 por cento das vezes. Qual é a taxa de erros de página máxima aceitável para um tempo de acesso efetivo de não mais do que 200 ns?

Resposta:

$$0{,}2\ \mu s = (1 - P) \times 0{,}1\ \mu s + (0{,}3P) \times 8\ ms + (0{,}7P) \times 20\ ms$$
$$0{,}1 = -0{,}1P + 2400\ P + 14000\ P$$
$$0{,}1 \cong 16.400\ P$$
$$P \cong 0{,}000006$$

15) [**] Quando ocorre um erro de página, o processo que está solicitando a página deve ser bloqueado enquanto espera que a página seja transferida do disco para a memória física. Suponha que exista um processo com cinco threads de nível de usuário e que o mapeamento de threads de usuário para threads do kernel seja muitos-para-um. Se um thread de usuário incorrer em um erro de página ao acessar sua pilha, os outros threads de usuário pertencentes ao mesmo processo também seriam afetados pelo erro de página — isto é, eles também teriam que esperar a página em erro ser transferida para a memória? Explique.

Resposta:

Sim, porque, como existe apenas um thread de kernel para todos os threads de usuário, esse thread de kernel é bloqueado enquanto espera que o erro de página seja resolvido. Como não existem outros threads de kernel para threads de usuário disponíveis, todos os outros threads de usuário no processo são afetados pelo erro de página.

16) [*] Suponha que você esteja monitorando a taxa segundo a qual o ponteiro do algoritmo do relógio (que indica a página candidata à substituição) se move. O que você pode dizer sobre o sistema se observar o seguinte comportamento:

a. O ponteiro está se movendo rapidamente.
b. O ponteiro está se movendo lentamente.

Resposta:

Se o ponteiro está se movendo rapidamente, então o programa está acessando um grande número de páginas simultaneamente. É muito provável que durante o período entre o momento em que o bit correspondente a uma página é desligado e verificado novamente a página seja acessada de novo e, portanto,

não possa ser substituída. Isso resultará em mais varreduras das páginas antes que uma página vítima seja encontrada. Se o ponteiro está se movendo lentamente, então o sistema de memória virtual está encontrando páginas candidatas a substituição de forma extremamente eficiente, indicando que muitas das páginas residentes não estão sendo acessadas.

17) Discuta situações (ou seja, crie exemplos hipotéticos) em que o algoritmo de substituição de páginas menos frequentemente utilizadas gere menos erros de página do que o algoritmo de substituição de páginas menos recentemente utilizadas. Discuta também sob que circunstâncias ocorre o oposto.

Resposta:

Considere a seguinte sequência de acessos à memória em um sistema que pode manter quatro páginas em memória: 1 1 2 3 4 5 1. Quando a página 5 for acessada, o algoritmo de substituição de páginas menos frequentemente utilizadas substituirá uma página diferente de 1 e, portanto, não incorrerá em erro de página quando a página 1 for acessada novamente. Entretanto, para a sequência “1 2 3 4 5 2”, o algoritmo da página menos recentemente utilizada funciona melhor.

18) Discuta situações (ou seja, crie exemplos hipotéticos) em que o algoritmo de substituição de páginas mais frequentemente utilizadas gere menos erros de página do que o algoritmo de substituição de páginas menos recentemente utilizadas. Discuta também sob que circunstâncias ocorre o oposto.

Resposta:

Considere a sequência a seguir em um sistema que mantém quatro páginas em memória: 1 2 3 4 4 4 5 1. O algoritmo de substituição de páginas mais frequentemente utilizadas expulsa a página 4 e busca a página 5, enquanto o algoritmo LRU expulsa a página 1. Isso não é provável que aconteça muito na prática. Para a sequência 1 2 3 4 4 4 5 1, o algoritmo LRU toma a decisão correta.

19) O sistema VAX/VMS usa um algoritmo de substituição FIFO para páginas residentes e um pool de quadros livres de páginas recentemente utilizadas. Suponha que o pool de quadros livres seja gerenciado usando a política de substituição de páginas menos recentemente utilizadas. Responda às seguintes perguntas:

a. Se ocorrer um erro de página e a página não existir no pool de quadros livres, como será gerado espaço livre para a página recém-solicitada?
b. Se ocorrer um erro de página e a página existir no pool de quadros livres, como o conjunto de páginas residentes e o pool de quadros livres serão gerenciados para gerar espaço para a página solicitada?
c. Para que o sistema degenera se o número de páginas residentes estiver posicionado em um?
d. Para que o sistema degenera se o número de páginas do pool de quadros livres for zero?

Resposta:

a. Quando ocorre um erro de página e a página não está no pool de quadros livres, então uma das páginas no pool de quadros livres é expulsa para disco, criando espaço para que uma das páginas residentes seja movida para o pool de quadros livres. A página acessada é então movida para o conjunto residente.
b. Quando ocorre um erro de página e a página existe no pool de quadros livres, então ela é movida para o conjunto de páginas residentes, enquanto uma das páginas residentes é movida para o pool de quadros livres.
c. Quando o número de páginas residentes é posicionado em 1, então o sistema degenera para o algoritmo de substituição de páginas utilizado no pool de quadros livres, o qual é tipicamente gerenciado ao estilo LRU.
d. Quando o número de páginas no pool de quadros livres é 0, então o sistema degenera para um algoritmo de substituição de páginas FIFO.

20) [***] Considere um sistema de paginação por demanda com as seguintes medidas de tempo de utilização:

Utilização da CPU = 20%
Disco de paginação = 97,7%
Outros dispositivos de I/O = 5%
Para cada uma das situações a seguir, diga se ela irá (ou poderá vir a) melhorar a utilização da CPU. Explique suas respostas.

a. Instalação de uma CPU mais rápida.
b. Instalação de um disco de paginação maior.
c. Aumento do grau de multiprogramação.
d. Diminuição do grau de multiprogramação.
e. Instalação de mais memória principal.
f. Instalação de um disco rígido mais rápido ou de múltiplos controladores com múltiplos discos rígidos.
g. Inclusão da pré-paginação nos algoritmos de busca de páginas.
h. Aumento do tamanho da página.

Resposta:

O sistema está obviamente gastando a maior parte do seu tempo paginando, indicando uma superalocação da memória. Se o nível de multiprogramação fosse reduzido, processos residentes causariam erros de página com menos frequência e a utilização da CPU melhoraria. Outra maneira de melhorar o desempenho seria obter mais memória física ou um disco de paginação mais rápido.

a. Instalação de uma CPU mais rápida – Não.
b. Instalação de um disco de paginação maior – Não.
c. Aumento do grau de multiprogramação – Não.
d. Diminuição do grau de multiprogramação – Sim.
e. Instalação de mais memória principal – Provavelmente para melhorar a utilização da CPU, já que mais páginas poderão permanecer residentes e não será exigida paginação para os discos ou a partir deles.
f. Instalação de um disco rígido mais rápido ou de múltiplos controladores com múltiplos discos rígidos – Também é uma melhoria, pois o gargalo no disco será removido, pela resposta mais rápida e mais throughput para os discos, e a CPU obterá mais dados mais rapidamente.
g. Inclusão da pré-paginação nos algoritmos de busca de páginas – Novamente a CPU obterá mais dados mais rapidamente e, assim, estará sendo mais utilizada. Esse é o caso apenas se a ação de paginação for receptiva à pré-busca (isto é, algum acesso é sequencial).
h. Aumento do tamanho da página – O aumento do tamanho da página resultará em menos erros de página se os dados estiverem sendo acessados sequencialmente. Se o acesso aos dados for mais ou menos aleatório, mais ação de paginação pode decorrer daí, porque menos páginas poderão ser mantidas na memória e mais dados serão transferidos por erros de página. Assim, é provável que essa mudança decresça a utilização da CPU em vez de aumentá-la.

21) [**] Suponha que uma máquina forneça instruções que podem acessar localizações de memória utilizando o esquema de endereçamento indireto de um nível. Que sequência de erros de página ocorrerá se todas as páginas de um programa estiverem não-residentes e a primeira instrução do programa for uma operação de carga de memória indireta? O que acontecerá se o sistema operacional estiver usando uma técnica de alocação de quadros por processo e somente duas páginas forem alocadas a este processo?

Resposta:

Os seguintes erros de página terão lugar: erro de página no acesso à instrução, erro de página no acesso à localização de memória que contém um ponteiro para a localização de memória alvo e erro de página quando

a localização de memória alvo for acessada. O sistema operacional gerará três erros de página, com a terceira página substituindo a página que contém a instrução. Se a instrução precisar ser buscada novamente para repetir a instrução capturada, então a sequência de erros de página continuará indefinidamente. Se a instrução for armazenada em cache em um registrador, então estará apta a executar completamente após o terceiro erro de página.

22) [*] Suponha que sua política de substituição (em um sistema paginado) seja examinar cada página regularmente e descartar a página se ela não tiver sido usada desde o último exame. O que você ganharia e o que perderia usando esta política em vez da substituição LRU ou da segunda chance?

Resposta:

Tal algoritmo pode ser implementado com o uso de um bit de referência. Após cada exame, o bit é posicionado como 0 e posicionado de volta como 1 se a página for referenciada. O algoritmo selecionaria então uma página arbitrária para substituição, do conjunto de páginas não utilizadas desde o último exame.

A vantagem de tal algoritmo é a sua simplicidade – nada mais do que um bit de referência precisa ser mantido. A desvantagem desse algoritmo é que ele ignora a localidade ao utilizar apenas um quadro de tempo curto para determinar se deve expulsar a página ou não. Por exemplo, uma página pode ser parte do conjunto ativo de um processo, mas pode ser expulsa porque não foi referenciada desde o último exame (isto é, nem todas as páginas no conjunto ativo podem ser referenciadas entre os exames).

23) Considere um sistema de paginação por demanda com um disco de paginação que tem um tempo médio de acesso e transferência de 20 milissegundos. Os endereços são traduzidos por intermédio de uma tabela de páginas na memória principal, com tempo de acesso de 1 microssegundo por acesso à memória. Assim, cada referência à memória, por intermédio da tabela de páginas, exige dois acessos. Para melhorar este tempo, adicionamos uma memória associativa que reduz o tempo de acesso para uma referência à memória se a entrada da tabela de páginas está na memória associativa.

Suponha que 80 por cento dos acessos ocorram à memória associativa e que, do restante, 10 por cento (ou 2 por cento do total) provoque erros de página. Qual é o tempo efetivo de acesso à memória?

Resposta:

$$EAT = (0{,}8 * 1\ \mu s) + (0{,}18 * 2\ \mu s) + (0{,}02 * 20002\ \mu s) = 401{,}2\ \mu s$$

24) [R] Qual é a causa da atividade improdutiva (*thrashing*)? Como o sistema detecta a atividade improdutiva? Uma vez que ela seja detectada, o que o sistema pode fazer para eliminar este problema?

Resposta:

A atividade improdutiva é causada por subalocação do número mínimo de páginas requerido por um processo, obrigando-o a erros de página contínuos. O sistema pode detectar a atividade improdutiva avaliando o nível de utilização da CPU comparado ao nível de multiprogramação. Ela pode ser eliminada reduzindo-se o nível de multiprogramação

25) [*] É possível que um processo tenha dois conjuntos ativos, um representando dados e outro representando código? Explique.

Resposta:

Sim. De fato muitos processadores fornecem dois TLBs exatamente por essa razão. Como exemplo, o código sendo acessado por um processo pode reter o mesmo conjunto ativo por um longo período de tempo. Entretanto, os dados que o código acessa podem mudar refletindo assim uma mudança no conjunto ativo para acessos aos dados.

Cap. 10: INTERFACE DO SISTEMA DE ARQUIVOS

1) [**] Alguns sistemas suportam muitos tipos de estruturas para os dados de um arquivo enquanto outros suportam simplesmente uma cadeia de bytes. Quais são as vantagens e desvantagens de cada abordagem?

Resposta:

Uma vantagem de ter o suporte do sistema para diferentes estruturas de arquivo é que o suporte vem do sistema; não são requeridas aplicações individuais para fornecer o suporte. Além disso, se o sistema oferecer suporte para diferentes estruturas de arquivo, ele poderá implementar o suporte presumivelmente de forma mais eficiente do que uma aplicação.

A desvantagem de o sistema fornecer suporte para tipos definidos de arquivos é que isso aumenta o tamanho do sistema. Além disso, aplicações que podem requerer diferentes tipos de arquivo, além do que é fornecido pelo sistema, podem não ser capazes de ser executadas em tais sistemas. Uma estratégia alternativa é o sistema operacional não definir suporte para estruturas de arquivos e tratar todos os arquivos como uma série de bytes. Essa é a abordagem escolhida pelos sistemas UNIX. A vantagem dessa abordagem é que ela simplifica o suporte do sistema operacional a sistemas de arquivos, já que o sistema não precisa mais fornecer a estrutura para diferentes tipos de arquivo. Além disso, ela permite que as aplicações definam estruturas de arquivos, aliviando assim a situação em que um sistema pode não fornecer uma definição de arquivo requerida por uma aplicação específica.

2) [R] Explique o objetivo das operações `open()` e `close()`.

Resposta:

O objetivo das operações open() e close() é:

- A operação open() informa ao sistema que o arquivo nomeado está para se tornar ativo.
- A operação close() informa ao sistema que o arquivo nomeado não está mais em uso ativo pelo usuário que emitiu a operação.

3) [*] Dê um exemplo de uma aplicação em que os dados de um arquivo devam ser acessados na seguinte ordem:
   a. Sequencialmente
   b. Aleatoriamente

Resposta:

Duas aplicações possíveis são:

a. Imprimir o conteúdo do arquivo
b. Imprimir o conteúdo do registro i. Esse registro pode ser encontrado utilizando-se técnicas de hash ou de indexação.

4) Considere um sistema de arquivos em que um arquivo possa ser removido e seu espaço em disco requisitado enquanto ainda existirem links para o arquivo. Que problemas podem ocorrer se um novo arquivo for criado na mesma área de armazenamento ou com o mesmo nome de caminho absoluto? Como esses problemas podem ser evitados?

Resposta:

Seja A1 o arquivo antigo e A2 o novo arquivo. O usuário que desejar acessar A1 por meio de um link existente acessará na verdade A2. Observe que a proteção de acesso para o arquivo A1 é utilizada preferencialmente à associada ao arquivo A2.

Este problema pode ser evitado garantindo-se que todos os links para um arquivo apagado sejam também apagados. Isso pode ser alcançado de diversas maneiras:

a. mantendo uma lista de todos os links para um arquivo, removendo cada um deles quando o arquivo for apagado
b. retendo os links, removendo-os quando for feita uma tentativa de acesso a um arquivo apagado
c. mantendo uma lista de referência ao arquivo (ou contador), apagando o arquivo somente depois que todos os links ou referências ao arquivo tiverem sido removidos.

5) [**] A tabela de arquivos abertos é utilizada para manter informações sobre os arquivos que estão correntemente abertos. O sistema operacional deveria manter uma tabela separada para cada usuário ou apenas uma tabela que contenha referências aos arquivos que estão sendo correntemente acessados por todos os usuários? Se o mesmo arquivo estiver sendo acessado por dois programas ou usuários diferentes, deveriam existir entradas separadas na tabela de arquivos abertos?

Resposta:

Ao manter uma tabela de arquivos abertos central, o sistema operacional pode executar a seguinte operação que seria impraticável de outro modo. Considere um arquivo que esteja sendo correntemente acessado por um ou mais processos. Se o arquivo for apagado, então ele não poderá ser removido do disco até que todos os processos que o estejam acessando o tenham fechado. Essa verificação pode ser executada somente se existir uma contagem centralizada do número de processos que estão acessando o arquivo. Por outro lado, se dois processos estiverem acessando o arquivo, então dois estados separados precisam ser mantidos para controlar a localização corrente das partes do arquivo que estão sendo acessadas pelos dois processos. Isso requer que o sistema operacional mantenha entradas separadas para os dois processos.

6) [***] Quais são as vantagens e desvantagens de fornecer locks obrigatórios em vez de locks consultivos cujo uso é deixado a critério dos usuários?

Resposta:

Em muitos casos, programas separados podem estar querendo tolerar o acesso concorrente a um arquivo sem requerer a necessidade de obter locks, garantindo assim a exclusão mútua para os arquivos. A exclusão mútua pode ser garantida por outras estruturas de programa, tais como locks de memória ou outras formas de sincronização. Em tais situações, os locks obrigatórios limitariam a flexibilidade da forma de acesso aos arquivos e poderiam também aumentar os overheads associados ao acesso a arquivos.

7) [**] Quais são as vantagens e desvantagens de registrar o nome do programa criador com os atributos do arquivo (como é feito no sistema operacional Macintosh)?

Resposta:

Ao registrar o nome do programa criador, o sistema operacional fica apto a implementar recursos (tal como a invocação automática do programa quando o arquivo é acessado) com base nessa informação. Isso, contudo, adiciona overhead ao sistema operacional e requer espaço no descritor de arquivos.

8) [**] Se o sistema operacional soubesse que uma determinada aplicação iria acessar os dados de um arquivo de modo sequencial, como ele poderia usar esta informação para melhorar o desempenho?

Resposta:

Quando um bloco é acessado, o sistema de arquivos pode executar uma pré-busca dos blocos subsequentes antecipando futuras solicitações desses blocos. Essa otimização de pré-busca reduziria o tempo de espera experimentado pelo processo em futuras solicitações.

9) [**] Dê um exemplo de uma aplicação que pudesse se beneficiar do suporte do sistema operacional ao acesso aleatório a arquivos indexados.

Resposta:

Uma aplicação que mantém um banco de dados de entradas pode se beneficiar de tal suporte. Por exemplo, se um programa estiver mantendo um banco de dados de estudantes, então os acessos ao banco de dados não podem ser modelados por qualquer padrão de acesso predeterminado. O acesso aos registros é aleatório, e a localização dos registros seria mais eficiente se o sistema operacional fornecesse alguma forma de indexação baseada em árvore.

Cap. 11: IMPLEMENTAÇÃO DO SISTEMA DE ARQUIVOS

1) [*] Por que o mapa de bits para alocação de arquivos deve ser mantido em memória de massa, e não na memória principal?

Resposta:

Em caso de queda do sistema (falha de memória), a lista de espaços livres não seria perdida como seria o caso se o mapa de bits tivesse sido armazenado na memória principal.

2) [***] Considere um sistema que suporte as estratégias de alocação contígua, encadeada e indexada. Que critérios devem ser levados em conta na decisão de qual estratégia é a melhor para um arquivo em particular?

Resposta:

- Contígua — se o arquivo é usualmente acessado sequencialmente e é relativamente pequeno.
- Encadeada — se o arquivo é grande e usualmente acessado sequencialmente.
- Indexada — se o arquivo é grande e usualmente acessado aleatoriamente.

3) [**] Um problema com a alocação contígua é que o usuário deve pré-alocar espaço suficiente para cada arquivo. Se o arquivo crescer a ponto de ficar maior do que o espaço a ele alocado, ações especiais devem ser levadas a efeito. Uma solução para este problema é definir uma estrutura de arquivo consistindo de uma área contígua inicial (de um tamanho especificado). Se essa área for preenchida, o sistema operacional definirá automaticamente uma área excedente que será encadeada na área contígua inicial. Se a área excedente for preenchida, outra área excedente será alocada. Compare esta implementação de um arquivo com as implementações padrão contígua e encadeada.

Resposta:

Esse método requer mais overhead do que a alocação contígua padrão e menos overhead do que a alocação encadeada padrão.

4) [*] Como os caches ajudam a melhorar o desempenho? Por que os sistemas não utilizam mais caches ou caches maiores se eles são tão úteis?

Resposta:

Os caches permitem que componentes de velocidades diferentes se comuniquem mais eficientemente armazenando, temporariamente, dados de um dispositivo mais lento em um dispositivo mais rápido (o cache). Os caches são, quase por definição, mais caros do que o dispositivo para o qual eles estão armazenando, e, assim, o aumento do número ou do tamanho dos caches aumenta o custo do sistema.

5) [***] Explique como a camada VFS permite que um sistema operacional suporte facilmente múltiplos tipos de sistemas de arquivos.

Resposta:

O VFS introduz uma camada de acesso indireto (por referências) na implementação do sistema de arquivos. De muitas formas, é semelhante às técnicas de programação orientada a objetos. As chamadas de sistema podem ser feitas genericamente (independentemente do tipo do sistema de arquivos). Cada tipo de sistema de arquivos fornece suas chamadas de sistema e estruturas de dados à camada VFS. Uma chamada de sistema é traduzida para as funções próprias específicas do sistema de arquivos alvo na camada VFS. O

programa chamador não tem código específico de sistema de arquivos, e as camadas superiores das estruturas das chamadas de sistema são igualmente independentes do sistema de arquivos. A tradução na camada VFS converte essas chamadas genéricas em operações específicas do sistema de arquivos.

6) [*] Considere um sistema de arquivos que utilize um esquema de alocação contígua modificado com suporte a extensões. Um arquivo é uma coleção de extensões, cada extensão correspondendo a um conjunto contíguo de blocos. Uma questão-chave em tais sistemas é o grau de variabilidade no tamanho das extensões. Quais são as vantagens e desvantagens dos seguintes esquemas?
   a. Todas as extensões são do mesmo tamanho e o tamanho é predeterminado.
   b. As extensões podem ser de qualquer tamanho e são alocadas dinamicamente.
   c. As extensões podem ser de poucos tamanhos fixos e esses tamanhos são predeterminados.

   Resposta:

Se todas as extensões são do mesmo tamanho e o tamanho é predeterminado, então o esquema de alocação de blocos ficará simplificado. Um simples mapa de bits ou lista de livres para extensões seria suficiente. Se as extensões podem ser de qualquer tamanho e são alocadas dinamicamente, então serão requeridos esquemas de alocação mais complexos. Pode ser difícil encontrar uma extensão do tamanho apropriado, e pode haver fragmentação externa. Pode ser utilizado o alocador do sistema de pares discutido nos capítulos anteriores para projetar um alocador apropriado. Quando as extensões podem ser de poucos tamanhos fixos e esses tamanhos são predeterminados, teria que ser mantido um mapa de bits ou lista de livres separados para cada tamanho possível. Esse esquema tem complexidade e flexibilidade intermediárias em comparação aos esquemas anteriores.

7) [**] Quais são as vantagens da variante da alocação encadeada que utiliza uma FAT para encadear os blocos de um arquivo?

   Resposta:

A vantagem é que enquanto se estiver acessando um bloco que esteja armazenado no meio de um arquivo, sua localização pode ser determinada procurando-se os ponteiros armazenados na FAT em oposição ao acesso de todos os blocos individuais do arquivo, de modo sequencial, para encontrar o ponteiro para o bloco alvo. Normalmente, a maior parte da FAT pode ser armazenada em cache na memória, e, portanto, os ponteiros podem ser determinados apenas com acessos à memória, em vez de exigirem acesso aos blocos em disco.

8) [**] Alguns sistemas de arquivos permitem que a memória em disco seja alocada em níveis de granularidade diferentes. Por exemplo, um sistema de arquivos poderia alocar 4 KB de espaço em disco como um único bloco de 4 KB ou como oito blocos de 512 bytes. Como poderíamos tirar vantagem dessa flexibilidade para melhorar o desempenho? Que modificações teriam de ser feitas no esquema de gerenciamento do espaço livre para suportar essa característica?

   Resposta:

Tal esquema diminuiria a fragmentação interna. Se um arquivo tem 5 KB, então ele pode ser alocado a um bloco de 4 KB e dois blocos de 512 bytes contíguos. Além de manter um mapa de bits de blocos livres, seria também necessário manter um estado extra com relação aos subblocos que estão sendo utilizados correntemente dentro de um bloco. O alocador teria então que examinar esse estado extra para alocar sub-blocos e fundi-los para obter o bloco maior quando todos os sub-blocos ficarem livres.

9) [*] Discuta como otimizações de desempenho para sistemas de arquivos podem resultar em dificuldades para manter a consistência dos sistemas em caso de quedas do computador.

Resposta:

A principal dificuldade que pode surgir se deve às atualizações retardadas de dados e metadados. As atualizações podem ser retardadas na esperança de que os mesmos dados possam ser atualizados no futuro ou de que os dados atualizados possam ser temporários e apagados no futuro próximo. Se porém o sistema caísse sem ter confirmado as atualizações retardadas, então a consistência do sistema de arquivos seria destruída.

10) [**] Explique por que o registro em log de atualizações de metadados garante a recuperação de um sistema de arquivos após esse sistema de arquivos cair.

Resposta:

Para que um sistema de arquivos seja recuperável após uma queda, ele deve ser consistente ou deve ser capaz de se tornar consistente. Portanto, temos que provar que o registro em log das atualizações de metadados mantém o sistema de arquivos em estado consistente ou passível de se tornar consistente. Para que um sistema de arquivos se torne inconsistente, os metadados devem ser gravados de forma incompleta ou na ordem errada nas estruturas de dados do sistema de arquivos. Com o registro em log dos metadados, as gravações são feitas em um log sequencial. A transação completa é gravada ali antes de ser movida para as estruturas do sistema de arquivos. Se o sistema cair durante as atualizações de dados no sistema de arquivos, essas poderão ser completadas com base nas informações do log. Assim, o registro em log garante que as mudanças no sistema de arquivos sejam feitas completamente (tanto antes como depois de uma queda). A ordem correta das alterações é garantida por causa das gravações sequenciais no log. Se uma alteração for feita de forma incompleta no log, ela será descartada, sem mudanças nas estruturas do sistema de arquivos. Portanto, as estruturas são consistentes ou podem se tornar consistentes de modo trivial por meio da reexecução do registro em log dos metadados.

HISTÓRICO DE VERSÕES

[2013-03-22] – v1.0.0

- Adição de exercícios dos Capítulos 10 e 11

[2013-02-14] – v0.9.0

- Adição de exercícios dos Capítulos 8 e 9
- Adicionadas classificações aos exercícios, conforme nota após o Sumário.

[2013-01-22] – v0.7.0

- Adição de exercícios dos Capítulos 6 e 7

[2012-12-08] – v0.5.0

- Versão Inicial. Capítulos 1 a 5