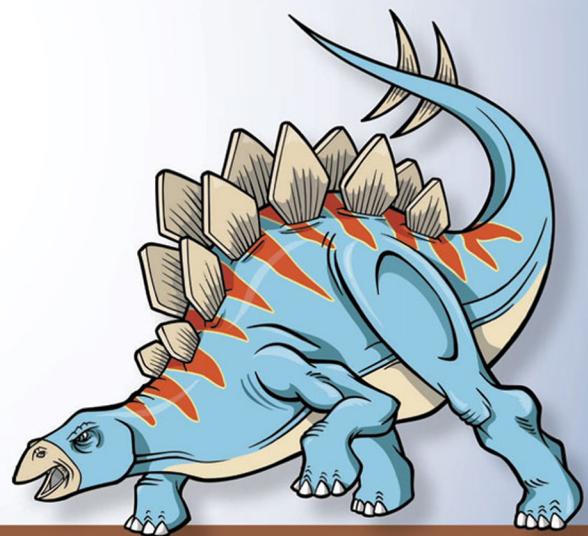


MATERIAL SUPLEMENTAR PARA ACOMPANHAR



FUNDAMENTOS DE SISTEMAS OPERACIONAIS

Abraham Silberschatz
Peter Baer Galvin
Greg Gagne



Nona Edição



Introdução

Exercícios Práticos

- 1.1 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, serve a duas funções principais: (1) supervisão da execução de programas de usuário para evitar erros e o uso impróprio do computador; (2) gerenciamento da operação e controle dos dispositivos de I/O.

- 1.2 Enfatizamos a necessidade de o sistema operacional usar eficientemente o hardware do computador. Quando é apropriado que o sistema operacional ignore esse princípio e “desperdice” recursos? Por que um sistema assim não está na verdade sendo ineficiente?

Resposta:

Sistemas monousuário devem maximizar o uso do sistema para o usuário. Uma GUI pode “desperdiçar” ciclos da CPU, mas otimiza a interação do usuário com o sistema.

- 1.3 Qual é a principal dificuldade que um programador deve superar ao escrever um sistema operacional para um ambiente de tempo real?

Resposta:

A principal dificuldade é manter o sistema operacional dentro das restrições de tempo fixadas para um sistema de tempo real. Se um sistema não concluir uma tarefa em determinado intervalo de tempo, pode causar uma paralisação do sistema inteiro que está em execução. Logo, ao escrever um sistema operacional para um sistema de tempo real, o programador deve se certificar de que seus esquemas de scheduling não permitam que o tempo de resposta exceda a restrição de tempo.

- 1.4 Lembrando-se das diversas definições de **sistema operacional**, considere se ele deve incluir aplicações como navegadores da web e programas de e-mail. Defenda tanto que ele deve como que ele não deve fazer isso, e fundamentalmente suas respostas.

Resposta:

Um argumento a favor da inclusão de aplicações populares no sistema operacional é que, se a aplicação estiver embutida no sistema operacional, provavelmente poderá se beneficiar melhor dos recursos do kernel e, portanto, terá vantagens de desempenho que não teria uma aplicação executada fora do kernel. No entanto, normalmente os argumentos contra a inclusão de aplicações no sistema operacional prevalecem: (1) aplicações são aplicações — não fazem parte do sistema operacional; (2) nenhum ganho de desempenho obtido com a execução dentro do kernel compensa as vulnerabilidades de segurança; (3) leva a um sistema operacional inchado.

- 1.5

Como a diferença entre a modalidade de kernel e a modalidade de usuário funciona como um tipo rudimentar de sistema de proteção (segurança)?

Resposta:

A diferença entre modalidade de kernel e modalidade de usuário fornece uma forma rudimentar de proteção da maneira a seguir. Certas instruções só podem ser executadas quando a CPU está em modalidade de kernel. Da mesma forma, dispositivos de hardware só podem ser acessados quando o programa está sendo executado em modalidade de kernel. O controle de quantas interrupções podem ser habilitadas ou desabilitadas também só é possível quando a CPU está em modalidade de kernel. Consequentemente, a CPU tem uma capacidade muito limitada quando executando em modalidade de usuário, o que reforça a proteção de recursos críticos.

- 1.6

Qual das instruções a seguir deve ser privilegiada?

- Configurar o valor do timer.
- Ler o relógio.
- Limpar a memória.
- Emitir uma instrução de exceção.
- Desativar interrupções.
- Modificar entradas na tabela de *status* de dispositivos.
- Passar de modalidade de usuário para a de kernel.
- Acessar dispositivo de I/O.

Resposta:

As instruções a seguir precisam ser privilegiadas: Posicionar o valor do timer, limpar a memória, desativar interrupções, modificar entradas na tabela de *status* de

dispositivos, acessar dispositivo de I/O. As outras podem ser executadas em modalidade de usuário.

- 1.7 Alguns computadores antigos protegiam o sistema operacional alocando-o a uma partição da memória que não podia ser modificada tanto pelo job do usuário quanto pelo próprio sistema operacional. Descreva duas dificuldades que você acha que poderiam surgir nesse esquema.

Resposta:

Os dados requeridos pelo sistema operacional (senhas, controles de acesso, informações de contabilidade, e assim por diante) teriam de ser armazenados em memória não protegida, ou passados através dela; sendo, portanto, acessíveis a usuários não autorizados.

- 1.8 Algumas CPUs fornecem mais de duas modalidades de operação. Cite dois usos possíveis para essas múltiplas modalidades.

Resposta:

Embora a maioria dos sistemas só faça distinção entre modalidades de usuário e de kernel, algumas CPUs suportam múltiplas modalidades. Múltiplas modalidades podem ser usadas para fornecer uma política de segurança mais refinada. Por exemplo, em vez de fazer a distinção apenas entre modalidade de usuário e de kernel, você poderia fazer a distinção entre diferentes tipos de modalidade de usuário. Talvez usuários pertencentes ao mesmo grupo pudessem executar os códigos uns dos outros. A máquina entraria em uma modalidade especificada quando um desses usuários estivesse executando código. Quando a máquina estivesse nessa modalidade, um membro do grupo poderia executar código pertencente a qualquer outro membro do grupo.

Outra possibilidade seria fornecer diferentes distinções dentro do código do kernel. Por exemplo, um modalidade específica poderia permitir que drivers de dispositivos USB fossem executados. Isso significaria que os dispositivos USB poderiam ser atendidos sem ter de passar para a modalidade de kernel, permitindo, essencialmente, que drivers de dispositivos USB fossem executados em uma quase modalidade de usuário/kernel.

- 1.9 Os timers podem ser usados para computar a hora corrente. Forneça uma breve descrição de como isso pode ser feito.

Resposta:

Um programa poderia usar a abordagem a seguir para computar a hora corrente utilizando interrupções de timer. O programa poderia posicionar um timer para algum momento futuro e adormecer. Quando fosse despertado pela interrupção, poderia atualizar seu estado local, que está sendo usado para rastrear o número de interrupções que recebeu até então. Em seguida, poderia repetir esse processo de posicionar continuamente inter-

rupções de timer e atualizar seu estado local quando as interrupções fossem realmente ativadas.

- 1.10 Cite duas razões que tornam os caches úteis. Que problemas eles resolvem? Que problemas causam? Se um cache puder ser tão grande quanto o dispositivo para o qual está armazenando (por exemplo, um cache tão extenso quanto um disco), por que não lhe dar esse tamanho e eliminar o dispositivo?

Resposta:

Os caches são úteis quando dois ou mais componentes precisam trocar dados, e esses componentes executam transferências com velocidades diferentes. Os caches resolvem o problema da transferência fornecendo um buffer de velocidade intermediária entre os componentes. Se o dispositivo rápido achar os dados de que precisa no cache, não precisará esperar pelo dispositivo mais lento. Os dados no cache devem ser mantidos consistentes com os dados nos componentes. Se um componente tiver um valor de dado alterado, e o dado também estiver no cache, o cache também deve ser atualizado. Isso é um problema principalmente em sistemas multiprocessadores em que mais de um processo pode estar acessando um dado. Um componente pode ser eliminado por um cache de mesmo tamanho, mas apenas se: (a) o cache e o componente tiverem capacidade equivalente de salvamento de estado (isto é, se o componente retiver seus dados quando a energia é removida, o cache também deverá reter dados) e (b) o cache puder ser custeado, porque uma memória mais rápida tende a ser mais cara.

- 1.11 Qual a diferença entre os modelos cliente-servidor e entre pares dos sistemas distribuídos?

Resposta:

O modelo cliente-servidor distingue claramente os papéis do cliente e do servidor. Nesse modelo, o cliente solicita serviços que são fornecidos pelo servidor. O modelo entre pares não tem esses papéis rígidos. Na verdade, todos os nós do sistema são considerados pares e, portanto, podem atuar como clientes *ou* servidores — ou ambos. Um nó pode solicitar um serviço a outro par ou pode fornecer esse serviço a outros pares do sistema.

Por exemplo, consideremos um sistema de nós que compartilhem receitas culinárias. No modelo cliente-servidor, todas as receitas são armazenadas no servidor. Se um cliente quiser acessar uma receita, deve solicitá-la ao servidor especificado. Com o uso do modelo entre pares, um nó pode solicitar a outros nós pares a receita especificada. O nó (ou talvez os nós) que tiver a receita solicitada poderá fornecê-la ao nó solicitante. Observe como cada par pode atuar como um cliente (ele pode solicitar receitas) e como um servidor (pode fornecer receitas).



Estruturas do Sistema Operacional

Exercícios Práticos

2.1 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.

2.2 Quais são as cinco principais atividades de um sistema operacional relacionadas com o gerenciamento de processos?

Resposta:

As cinco atividades principais são:

- A criação e destruição tanto de processos de usuário quanto de sistema.
- A suspensão e retomada de processos.
- O fornecimento de mecanismos para sincronização de processos.
- O fornecimento de mecanismos para comunicação entre processos.
- O fornecimento de mecanismos para manipulação de deadlocks.

2.3 Quais são as três principais atividades de um sistema operacional relacionadas com o gerenciamento de memória?

Resposta:

As três atividades principais são:

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

2.4 Quais são as três principais atividades de um sistema operacional relacionadas com o gerenciamento de memória secundária?

Resposta:

As três atividades principais são:

- Gerenciamento do espaço livre.
- Alocação de memória.
- Scheduling de disco.

2.5 Qual é a finalidade do interpretador de comandos? Por que geralmente ele é separado do kernel?

Resposta:

Ele lê comandos do usuário ou de um arquivo de comandos e os executa, colocando-os, usualmente, em uma ou mais chamadas de sistema. Não é, em geral, parte do kernel, já que o interpretador de comandos é sujeito a modificações.

2.6 Que chamadas de sistema têm de ser executadas por um shell ou interpretador de comandos para iniciar um novo processo?

Resposta:

Em sistemas UNIX, uma chamada de sistema *fork*, seguida por uma chamada de sistema *exec*, precisa ser executada para iniciar um novo processo. A chamada *fork* clona o processo em execução corrente, enquanto a chamada *exec* substitui o processo que fez a chamada por um novo processo com um executável diferente.

2.7 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.

2.8 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.

2.9 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 que programas de nível de usuário fornecessem 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 confiáveis para acessar somente os dispositivos que eles podem acessar e acessá-los somente quando eles não estiverem sendo usados.
- c. **Manipulação do sistema de arquivos.** Existem muitos detalhes na criação, exclusão, alocação e nomeação de arquivos que os usuários não devem executar. Blocos de espaço em disco são utilizados por arquivos e devem ser formatados. 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 nem podem garantir aderência aos métodos de proteção nem 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 componham pacotes de informação, sejam enviadas ao controlador da rede, transmitidas por 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 ordenar 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 assegurar que eles não mudaram desde que foram gravados na mí-

dia. 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.

- 2.10** Por que alguns sistemas armazenam o sistema operacional em firmware enquanto outros o armazenam em disco?

Resposta:

Para certos dispositivos, tais como PDAs móveis e telefones celulares, um disco com um sistema de arquivos pode não estar disponível para o dispositivo. Nessa situação, o sistema operacional deve estar armazenado em firmware.

- 2.11** Como seria o projeto de um sistema que permitisse a escolha, entre sistemas operacionais, daquele a ser inicializado? O que o programa bootstrap teria que fazer?

Resposta:

Considere um sistema que queira 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 locações 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 a serem inicializados; eles também são normalmente projetados para inicializar um sistema operacional default se nenhuma escolha for feita pelo usuário.



Processos

Exercícios Práticos

- 3.1 Utilizando o programa mostrado na Figura 3.30, explique qual será a saída na Linha A.

Resposta:

O resultado ainda é 5, já que o filho atualiza sua cópia de value. Quando o controle retornar ao pai, sua variável value continuará sendo 5.

- 3.2 Incluindo o processo-pai inicial, quantos processos são criados pelo programa mostrado na Figura 3.31?

Resposta:

São criados 16 processos.

- 3.3 As versões originais do sistema operacional móvel iOS da Apple não forneciam meios de processamento concorrente. Discuta três grandes complicações que o processamento concorrente adiciona a um sistema operacional.

- 3.4 O processador UltraSPARC da Sun tem múltiplos conjuntos de registradores. Descreva o que acontece quando ocorre uma mudança de contexto, e o novo contexto já está carregado em um dos conjuntos de registradores. O que acontece quando o novo contexto está na memória, e não em um conjunto de registradores, e todos os conjuntos de registradores estão sendo usados?

Resposta:

O ponteiro do conjunto de registradores correntes da CPU é alterado para apontar para o conjunto que contém o novo contexto, o que leva muito pouco tempo. Se o contexto estiver em memória, um dos contextos em um conjunto de registradores deverá ser selecionado e movido para a memória, e o novo contexto deverá ser carregado da memória, no conjunto de registradores. Esse processo leva um pouco mais de tempo do que em sistemas com um conjunto de registradores, dependendo de como a vítima da substituição é selecionada.

- 3.5 Quando um processo cria um novo processo usando a operação fork(), qual dos estados a seguir é compartilhado entre o processo-pai e o processo-filho?

- Pilha
- Heap
- Segmentos de memória compartilhada

Resposta:

Somente os segmentos de memória compartilhada são compartilhados entre o processo-pai e o processo-filho

recém-criado por fork. São feitas cópias da pilha e do heap para o processo recém-criado.

- 3.6 No que diz respeito ao mecanismo RPC, considere a semântica “exatamente um”. O algoritmo para implementação dessa semântica é executado corretamente, mesmo quando a mensagem ACK retornada ao cliente é perdida por causa de um problema na rede? Descreva a sequência de mensagens e discuta se a semântica “exatamente um” continua sendo preservada.

Resposta:

A semântica “exatamente um” garante que um procedimento remoto seja executado exatamente uma vez e somente uma vez. O algoritmo geral para fornecer essa garantia combina um esquema de conhecimento (ACK — *acknowledgment*) com marcadores de tempo (ou algum outro contador incremental que permita ao servidor distinguir entre mensagens duplicadas).

A estratégia geral é que o cliente envie uma RPC ao servidor junto com um marcador de tempo. O cliente também dará início a um relógio de expiração de tempo. O cliente então esperará por uma de duas ocorrências: (1) ele receberá um ACK do servidor indicando que o procedimento remoto foi executado, ou (2) ele expirará por tempo. Se o cliente expirar por tempo, assumirá que o servidor foi incapaz de executar o procedimento remoto; assim, o cliente invocará a RPC uma segunda vez, enviando um último marcador de tempo. O cliente pode não receber o ACK por uma de duas razões: (1) a RPC original jamais foi recebida pelo servidor, ou (2) a RPC foi corretamente recebida — e executada pelo servidor —, mas o ACK foi perdido. Na situação (1), o uso de ACKs permitirá que o servidor finalmente receba e execute a RPC. Na situação (2), o servidor receberá uma RPC em duplicata e utilizará o marcador de tempo para identificá-la como uma duplicata e não executar a RPC uma segunda vez. É importante observar que o servidor deverá enviar um segundo ACK de volta ao cliente para informá-lo de que a RPC foi executada.

- 3.7 Suponha que um sistema distribuído seja suscetível a falhas no servidor. Que mecanismos seriam necessários para garantir a semântica “exatamente um” na execução de RPCs?

Resposta:

O servidor deverá rastrear em memória estável (tal como um log em disco) informações referentes a quais opera-

ções de RPC foram recebidas, se elas foram executadas com sucesso, e os resultados associados às operações. Quando ocorre uma queda do servidor e é recebida uma

mensagem RPC, o servidor pode verificar se a RPC foi anteriormente executada e garantir, portanto, a semântica “exatamente um” para a execução de RPCs.

Threads



Exercícios Práticos

- 4.1 Forneça dois exemplos de programação em que a criação de múltiplos threads proporcione melhor desempenho do que uma solução com um único thread.
- Resposta:**
- Um servidor web que sirva cada solicitação em um thread separado.
 - Uma aplicação paralelizada, tal como uma multiplicação de matrizes, em que diferentes partes da matriz podem ser trabalhadas em paralelo.
 - 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.
- 4.2 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:**
- Threads de nível de usuário são desconhecidos pelo kernel, enquanto o kernel está ciente dos threads de nível de kernel.
 - 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.
 - 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.
- 4.3 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.

- 4.4 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 um conjunto de registradores, a pilha e as prioridades.

- 4.5 Suponha que um sistema operacional mapeie threads de nível de usuário para o kernel usando o modelo muitos-para-muitos e que o mapeamento seja feito por meio de LWPs. Além disso, o sistema permite que os desenvolvedores criem threads de tempo real para uso em sistemas de tempo real. É necessário vincular um thread de tempo real a um LWP? Explique.

Resposta:

Sim. O timing é crucial para as aplicações de tempo real. Se um thread é marcado como de tempo real mas não está vinculado a um LWP, o thread pode ter que esperar até que seja ligado a um LWP antes da execução. Suponha que um thread de tempo real esteja em execução (vinculado a um LWP) e, então, seja bloqueado (isto é, deve executar I/O, sofreu preempção por um thread de tempo real de prioridade mais alta, está esperando por um lock de exclusão mútua etc.). Enquanto o thread de tempo real estiver bloqueado, o LWP ao qual ele estava vinculado foi atribuído a outro thread. Quando o thread de tempo real foi alocado ao schedule para nova execução, ele deverá primeiro esperar para ser vinculado a um LWP. Vinculando um LWP a um thread de tempo real, você está garantindo que o thread será capaz de entrar em execução, com demora mínima, assim que for alocado ao schedule.



Sincronização de Processos

Exercícios Práticos

- 5.1 Na Seção 5.4, mencionamos que a desabilitação de interrupções pode, com frequência, afetar o relógio do sistema. Explique por que isso pode ocorrer e como esses efeitos podem ser minimizados.

Resposta:

O relógio do sistema é atualizado sempre que é interrompido. Se as interrupções forem desabilitadas — particularmente por um longo período de tempo —, é possível que o relógio do sistema possa perder a hora certa facilmente. O relógio do sistema também é utilizado para fins de scheduling. Por exemplo, o quantum de tempo de um processo é expresso por um número de tiques do relógio. A cada interrupção do relógio, o scheduler determina se o quantum de tempo do processo em execução corrente expirou. Se as interrupções do relógio forem desabilitadas, o scheduler não poderá atribuir, com precisão, os quanta de tempo. Esse efeito pode ser minimizado desabilitando as interrupções do relógio somente por períodos muito curtos.

- 5.2 Explique por que o Windows, o Linux e o Solaris implementam múltiplos mecanismos de trancamento. Descreva as circunstâncias em que eles usam spinlocks, locks mutexes, semáforos, locks mutexes adaptativos e variáveis de condição. Em cada caso, explique por que o mecanismo é necessário.

Resposta:

Esses sistemas operacionais fornecem diferentes mecanismos de trancamento, dependendo das necessidades dos desenvolvedores das aplicações. Os spinlocks são úteis para sistemas multiprocessadores em que um thread pode executar em um busy-loop1 (por um curto período de tempo) em vez de incorrer no overhead de ser colocado em uma fila de adormecidos. Os mutexes são úteis para recursos de trancamento. O Solaris 2 utiliza mutexes adaptativos, significando que o mutex é implementado com um spinlock em máquinas multiprocessadoras. Os semáforos e as variáveis de condição são ferramentas mais apropriadas para sincronização, no caso em que um recurso deve ser mantido por um longo período de tempo, já que o spinning é ineficiente para uma longa duração.

- 5.3 Qual é o significado do termo **espera em ação**? Que outros tipos de espera existem em um sistema operacional? A espera em ação pode ser totalmente evitada? Explique sua resposta.

Resposta:

Espera em ação significa que um processo está esperando que uma condição seja satisfeita em um loop restrito, sem abando-

nar o processador. Alternativamente, um processo poderia esperar abandonando o processador, ficando bloqueado em uma condição e esperando ser desperto em algum momento apropriado no futuro. A espera em ação pode ser evitada, mas isso incorre no overhead associado à colocação de um processo para dormir e ter de despertá-lo quando o estado apropriado do programa for alcançado.

- 5.4 Explique por que os spinlocks não são apropriados para sistemas uniprocessadores, mas são usados com frequência em sistemas multiprocessadores.

Resposta:

Os spinlocks não são apropriados para sistemas de processador único porque a condição que removeria um processo do spinlock só pode ser obtida pela execução de um processo diferente. Se o processo não abandonar o processador, outros processos não terão a oportunidade de posicionar a condição do programa requerida para o primeiro processo avançar. Em um sistema multiprocessador, outros processos são executados em outros processadores e, portanto, modificam o estado do programa para liberar o primeiro processo do spinlock.

- 5.5 Mostre 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 a um semáforo. Se duas operações `wait` forem executadas em um semáforo quando seu valor for 1, se as duas operações não forem executadas atomicamente, é possível que ambas decrementem o valor do semáforo, violando assim a exclusão mútua.

- 5.6 Mostre como um semáforo binário pode ser usado para implementar a exclusão mútua entre n processos.

Resposta:

Os n processos compartilham um semáforo, mutex, inicializado com 1. Cada processo P_i é organizado como mostrado a seguir:

```
do {  
    wait (mutex);  
  
    /* seção crítica */  
  
    signal (mutex);  
  
    /* seção restante */  
}while (true);
```



Scheduling da CPU

Exercícios Práticos

- 6.1 Um algoritmo de scheduling da CPU determina uma ordem para a execução dos processos a serem alocados à CPU. Dados n processos a serem alocados a um processador, quantos schedules diferentes são possíveis? Forneça uma fórmula em função de n .

Resposta:

$$n! \quad (n \text{ factorial} = n \times n - 1 \times n - 2 \times \dots \times 2 \times 1).$$

- 6.2 Explique a diferença entre scheduling preemptivo e não preemptivo.

Resposta:

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

- 6.3 Suponha que os processos a seguir cheguem para execução nos momentos indicados. Cada processo será executado durante o período de tempo listado. Ao responder às perguntas, use o scheduling não preemptivo 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

- Qual é o tempo médio de turnaround desses processos com o algoritmo de scheduling FCFS?
- Qual é o tempo médio de turnaround desses processos com o algoritmo de scheduling SJF?
- O algoritmo SJF deveria melhorar o desempenho, mas observe que optamos por executar o processo P_1 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 scheduling 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 scheduling de conhecimento futuro.

Resposta:

- 10,53
- 9,53
- 6,86

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

- 6.4 Qual a vantagem de termos tamanhos diferentes para o quantum de tempo em níveis distintos de um sistema de enfileiramento multinível?

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.

- 6.5 Muitos algoritmos de scheduling da CPU são parametrizados. Por exemplo, o algoritmo RR requer um parâmetro que indique a parcela de tempo. Filas multiníveis com retroalimentação requerem parâmetros que definem o número de filas, o algoritmo de scheduling 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?

- Por prioridades e SJF
- Filas multiníveis com retroalimentação e FCFS
- Por prioridades e FCFS
- RR e SJF

Resposta:

- O job mais curto tem a prioridade mais alta.
- O nível mais baixo do MLFQ (filas multiníveis com retroalimentação) é o FCFS.

- c. O FCFS dá a prioridade mais alta ao job com o tempo de existência mais longo.
- d. Nenhuma.
- 6.6** Suponha que um algoritmo de scheduling (do nível do scheduling de CPU de curto prazo) favoreça 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:**
O algoritmo 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.
- 6.7** Explique a diferença entre o scheduling PCS e SCS.
- Resposta:**
O scheduling PCS é realizado localmente para o processo. É a forma como a biblioteca de threads organiza o schedule dos threads em LWPs disponíveis. O scheduling SCS corresponde à situação em que o sistema operacional organiza o schedule dos threads de kernel. Em sistemas que utilizam tanto o esquema muitos-para-um quanto o esquema muitos-para-muitos, os dois modelos de scheduling são fundamentalmente diferentes. Em sistemas que utilizam um-para-um, o PCS e o SCS são a mesma coisa.
- 6.8** Suponha que um sistema operacional mapeie threads de nível de usuário para o kernel usando o modelo muitos-para-muitos e que o mapeamento seja feito por meio 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 schedule. 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 schedule.

- 6.9** O scheduler tradicional do UNIX impõe um relacionamento inverso entre números de prioridade e prioridades: quanto mais alto o número, menor a prioridade. O scheduler recalcula as prioridades dos processos uma vez por segundo usando a função a seguir:

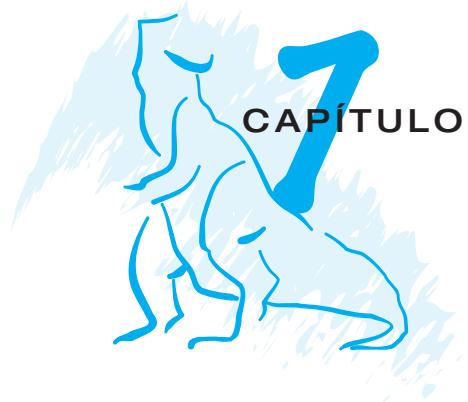
$$\text{Prioridade} = (\text{uso recente da CPU} / 2) + \text{base}$$

em que base = 60, e *uso recente da CPU* se refere a um valor indicando a frequência com que um processo usou a CPU desde que as prioridades foram recalculadas pela última vez.

Suponha que o uso recente da CPU pelo processo P_1 seja igual a 40, pelo processo P_2 seja igual a 18 e pelo processo P_3 seja igual a 10. Quais serão as novas prioridades desses três processos quando as prioridades forem recalculadas? Com base nessas informações, o scheduler tradicional do UNIX elevará ou rebaixará a prioridade relativa de um processo limitado por CPU?

Resposta:

As prioridades atribuídas aos processos são 80, 69 e 65, respectivamente. O scheduler diminui a prioridade relativa de processos limitados por CPU.



Deadlocks

Exercícios Práticos

- 7.1 Liste três exemplos de deadlocks que não estejam relacionados com 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.

- 7.2 Suponha que um sistema esteja em um estado inseguro. Demonstre ser possível que os processos concluam suas execuções 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:

Processo	Máx	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 porque o processo P_1 pode terminar liberando, assim, um total de quatro recursos, mas não podemos garantir que os processos P_0 e P_2 possam terminar. Entretanto, é possível que um processo libere recursos antes de solicitar qualquer outro. Por exemplo, o processo P_2 pode liberar um recurso, aumentando, assim, o número total de recursos para cinco. Isso permite que o processo P_0 termine, o que causa a liberação de um total de nove recursos, permitindo que o processo P_2 também termine.

- 7.3 Considere o seguinte instantâneo de um sistema:

Processo	Alocação				Max	Disponível
	A	B	C	D		
P_0	0	0	1	2	0	0
P_1	1	0	0	0	7	5
P_2	1	3	5	4	2	3
P_3	0	6	3	2	6	5
P_4	0	0	1	4	6	5

Responda às perguntas a seguir usando o algoritmo do banqueiro:

- Qual é o conteúdo da matriz **Necessidade**?
- O sistema está em estado de segurança?
- Se uma solicitação na forma (0,4,2,0) for feita pelo processo P_1 , ela poderá ser atendida imediatamente?

Resposta:

- O valor de **Necessidade** para os processos P_0 a P_4 são, respectivamente, (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0) e (0, 6, 4, 2).
- O sistema está em estado de segurança? Sim. Com **Disponível** sendo igual a (1, 5, 2, 0), tanto P_0 como P_3 poderia ser executado. Uma vez que o processo P_3 seja executado, liberará seus recursos, o que permitirá que todos os outros processos existentes sejam executados.
- A solicitação pode ser atendida imediatamente? Isso resulta no valor de **Disponível** sendo (1, 1, 0, 0). Uma ordem dos processos que podem terminar é P_0 , P_2 , P_3 , P_1 e P_4 .

- 7.4 Um método possível para a prevenção de deadlocks é contar com um recurso individual de mais alta ordem que deva ser solicitado antes de qualquer outro recurso. Por exemplo, se múltiplos threads tentarem acessar os objetos de sincronização $A \dots E$, pode ocorrer um deadlock. (Esses objetos de sincronização podem incluir mutexes, semáforos, variáveis de condição, e assemelhados). Podemos prevenir a ocorrência do deadlock adicionando um sexto objeto F . Sempre que um thread quiser adquirir o lock de sincronização de qualquer objeto $A \dots E$, antes deve adquirir o lock do objeto F . Essa solução é conhecida como **contentão**: os locks dos objetos $A \dots E$ estão contidos dentro do lock do objeto F . Compare esse esquema com o esquema de espera circular da Seção 7.4.4.

Resposta:

Essa não é provavelmente uma boa solução porque produz um escopo muito amplo. É melhor definir uma política de trancamento com um escopo o mais limitado possível.

- 7.5 Prove que o algoritmo de segurança apresentado na Seção 7.5.3 requer uma ordem de $m \times n^2$ operações.

Resposta:

A Figura 7.1 fornece código Java que implementa o algoritmo de segurança do algoritmo do banqueiro (a implementação completa do algoritmo do banqueiro está disponível com o download do código-fonte no site da LTC Editora).

```

for (int i = 0; i < n; i++) {
    // encontra primeiro um thread que possa
    terminar
    for (int j = 0; j < n; j++) {
        if (!finish[j]) {
            boolean temp = true;
            for (int k = 0; k < m; k++) {
                if (need[j][k] > work[k])
                    temp = false;
            }

            if (temp) { // se esse thread pode
            terminar
                finish[j] = true;
                for (int x = 0; x < m; x++)
                    work[x] += work[j][x];
            }
        }
    }
}

```

Figura 7.1 Algoritmo de segurança do algoritmo do banqueiro.

Como pode ser visto, os loops externos aninhados — os dois com n ciclos — fornecem desempenho de n^2 . Dentro desses loops externos estão dois loops internos sequenciais com m ciclos. A big-oh desse algoritmo é, então, $O(m \times n^2)$.

- 7.6 Considere um sistema de computação que executa 5.000 jobs por mês e não tem esquema de prevenção ou de impedimento de deadlocks. Os deadlocks ocorrem aproximadamente duas vezes por mês, e o operador deve encerrar e reexecutar cerca de 10 jobs por deadlock. Cada job custa perto de 2 dólares (em tempo de CPU), e os jobs encerrados tendem a ser executados até a metade quando são abortados.

Um programador de sistemas estimou que um algoritmo de impedimento de deadlocks (como o algoritmo do banqueiro) poderia ser instalado no sistema com um aumento de cerca de 10 por cento no tempo médio de execução por job. Estando a máquina, correntemente, com 30 por cento do tempo ocioso, todos os 5.000 jobs por mês poderiam continuar em execução, embora o tempo de turnaround aumentasse em média cerca de 20 por cento.

- Quais são os argumentos para a instalação do algoritmo de impedimento de deadlocks?
- Quais são os argumentos contra a instalação do algoritmo de impedimento de deadlocks?

Resposta:

Um argumento favorável à instalação do algoritmo de impedimento de deadlocks no sistema é que poderíamos garantir a não ocorrência de deadlocks. Além disso, apesar do aumento do tempo de turnaround, todos os 5.000 jobs poderiam ainda ser executados.

Um argumento contra a instalação de software de impedimento de deadlocks é que deadlocks ocorrem com pouca frequência e custam pouco quando ocorrem.

- 7.7 Um sistema pode detectar que algum de seus processos está sofrendo de 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 os fins dessa 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 não 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 P_a esteve esperando pelo recurso X por mais tempo do que o processo P_b , a solicitação do processo P_b seria adiada até que a solicitação do processo P_a 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.

- 7.8 Considere a política de alocação de recursos a seguir. Solicitações e liberações de recursos são permitidas a qualquer momento. Se uma solicitação de recursos não pode ser atendida porque os recursos não estão disponíveis, verificamos quaisquer processos que estejam bloqueados esperando por recursos. Se um processo bloqueado tem os recursos desejados, esses recursos são dele retirados e passados ao processo solicitante. O vetor de recursos pelos quais o processo bloqueado está esperando é aumentado para incluir os recursos que foram removidos.

Por exemplo, considere um sistema com três tipos de recursos e o vetor *Disponível* inicializado com (4,2,2). Se o processo P_0 solicita (2,2,1), ele os recebe. Se P_1 solicita (1,0,1), ele os recebe. Em seguida, se P_0 solicita (0,0,1), ele é bloqueado (recurso não disponível). Se P_2 solicitar agora (2,0,0), ele recebe o recurso disponível (1,0,0) e um recurso que estava alocado a P_0 (já que P_0 está bloqueado). O vetor *Alocação* de P_0 diminui para (1,2,1), e seu vetor *Necessidade* aumenta para (1,0,1).

- Pode ocorrer um deadlock? Se você responder “sim”, dê um exemplo. Se responder “não”, especifique que condição necessária não pode ocorrer.
- Pode ocorrer um bloqueio indefinido? Explique sua resposta.

Resposta:

- a. O deadlock não pode ocorrer porque existe preempção.
- b. Sim. Um processo não poderá jamais adquirir todos os recursos de que precisa se eles sofrerem preempção continuamente por uma série de solicitações, tais como as do processo C .
- 7.9 Suponha que você tenha codificado o algoritmo de segurança de impedimento de deadlocks e, agora, tenha sido solicitado a implementar o algoritmo de detecção de deadlocks. Pode fazer isso simplesmente usando o código do algoritmo de segurança e redefinindo $\text{Max}[i] = \text{Espera}[i] + \text{Alocação}[i]$, em que $\text{Espera}[i]$ é um vetor que especifica os recursos pelos quais o processo i está esperando, e

$\text{Alocação}[i]$ segue o que foi definido na Seção 7.5? Explique sua resposta.

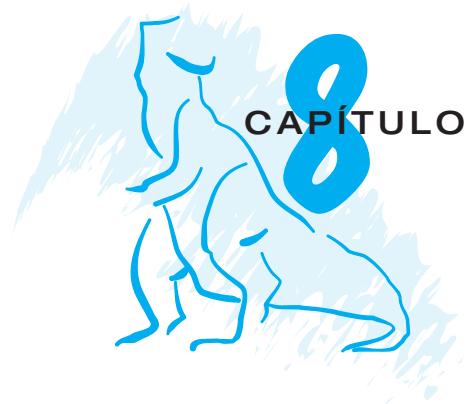
Resposta:

Sim. O vetor Max representa o número máximo de solicitações que um processo pode fazer. Quando calculamos o algoritmo de segurança, utilizamos a matriz Necessidade , que representa $\text{Max} - \text{Alocação}$. Outra maneira de pensar sobre isso é $\text{Max} = \text{Necessidade} + \text{Alocação}$. De acordo com a questão, a matriz Espera tem um papel semelhante ao da matriz Necessidade ; portanto, $\text{Max} = \text{Espera} + \text{Alocação}$.

- 7.10 É possível ocorrer um deadlock envolvendo apenas um processo com um único thread? Explique sua resposta.

Resposta:

Não. Isso deriva diretamente da condição de manter-e Esperar



Memória Principal

Exercícios Práticos

- 8.1 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 em um 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.

- 8.2 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-límite são fornecidos: um para instruções e outro para dados. O par de registradores base-límite 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 os processos 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.

- 8.3 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.

- 8.4 Considere um espaço de endereçamento lógico de 64 páginas com 1024 palavras cada, mapeado para uma memória física de 32 quadros.

- Quantos bits há no endereço lógico?
- Quantos bits há no endereço físico?

Resposta:

- Endereço lógico: 16 bits
- Endereço físico: 15 bits

- 8.5 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 um grande montante de memória de um local para outro. Que efeito a atualização de algum byte em uma página teria sobre a 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 locação 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.

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

- 8.7 O compartilhamento de segmentos entre processos sem a exigência de que eles tenham o mesmo número de segmento é possível em um sistema de segmentação vinculado dinamicamente.

- a. Defina um sistema que permita a vinculação estática e o compartilhamento de segmentos sem demandar que os números de segmentos sejam iguais.
- b. Descreva um esquema de paginação que permita que as páginas sejam compartilhadas sem requerer que os números das páginas sejam iguais.

Resposta:

Os dois problemas reduzem-se a um programa capaz de referenciar tanto seu próprio código como seus dados sem saber o número do segmento ou da página associado ao endereço. O MULTICS resolveu esse problema associando quatro registradores a cada processo. Um dos registradores tinha o endereço do segmento corrente do programa, outro tinha o endereço base da pilha, outro tinha o endereço base dos dados globais, e assim por diante. A ideia é que todas as referências tenham que ser indiretas, por intermédio de um registrador que mapeie para o segmento corrente ou o número da página. Alterando esses registradores, o mesmo código poderá ser executado por processos diferentes sem a mesma página ou os mesmos números de segmento.

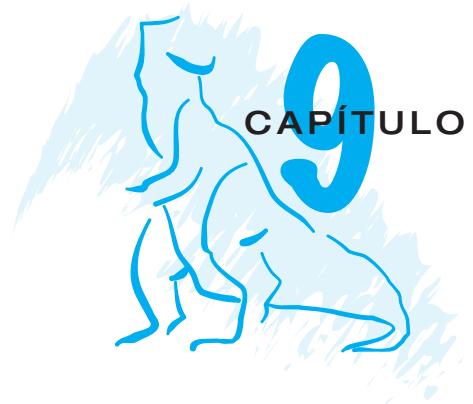
- 8.8** No IBM/370, a proteção à memória é fornecida por intermédio do uso de *chaves*. Uma chave é um valor de 4 bits. Cada bloco de memória de 2 K tem uma chave (a chave de armazenamento) associada a ele. A CPU também tem uma chave (a chave de proteção) associada a ela. Uma operação de armazenamento só é permitida quando as

duas chaves são iguais ou quando seu valor é zero. Quais dos esquemas de gerenciamento de memória a seguir poderiam ser usados com sucesso com esse hardware?

- a. Machine crua
- b. Sistema monousuário
- c. Multiprogramação com um número fixo de processos
- d. Multiprogramação com um número variável de processos
- e. Paginação
- f. Segmentação

Resposta:

- a. Não é necessário proteção; posicione a chave do sistema em 0.
- b. Posicione a chave do sistema em 0 quando em modalidade de supervisor.
- c. Os tamanhos das regiões devem ser fixados em incrementos de 2k bytes; aloque a chave com blocos de memória.
- d. O mesmo anterior.
- e. Os tamanhos dos quadros devem ser em incrementos de 2k bytes; aloque a chave com páginas.
- f. Os tamanhos dos segmentos devem ser em incrementos de 2k bytes; aloque a chave com segmentos.



Memória Virtual

Exercícios Práticos

- 9.1** 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 para o quadro livre. Após o término do I/O, a tabela de processos e a tabela de páginas são atualizadas e a instrução é reiniciada.

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

- O que é um limite inferior para o número de erros de página?
- O que é um limite superior para o número de erros de página?

Resposta:

- n
- p

- 9.3** Considere a tabela de páginas mostrada na Figura 9.30 para um sistema com endereços virtuais e físicos de 12 bits e páginas de 256 bytes. Na lista de quadros de páginas livres temos D , E e F (isto é, D é a cabeça da lista, E é o segundo, e F é o último).

Converta os endereços virtuais, a seguir, nos endereços físicos equivalentes em hexadecimais. Todos os números usam o formato hexadecimal. (Um travessão para um quadro de página indica que a página não está na memória.)

- 9EF
- 111
- 700
- 0FF

Resposta:

- 9EF – 0EF
- 111 – 211

- 700 – D00

- 0FF – EFF

- 9.4** Considere os algoritmos de substituição de páginas a seguir. Classifique esses algoritmos em uma escala de cinco pontos, de “ruim” a “perfeito”, de acordo com sua taxa de erros de página. Separe os algoritmos afetados pela anomalia de Belady daqueles que não o são.
- Substituição LRU
 - Substituição FIFO
 - Substituição ótima
 - Substituição da segunda chance

Resposta:

Classificação	Algoritmo	Sofre da anomalia de Belady
1	Ótimo	Não
2	LRU	Não
3	Segunda chance	Sim
4	FIFO	Sim

- 9.5** Discuta o suporte de hardware requerido pelo suporte à paginação por demanda.

Resposta:

Para cada operação de acesso à memória, a tabela de páginas precisa ser consultada para verificar se a página correspondente é residente ou não, e se o programa tem privilégios de leitura ou gravação para acessar a página. Essas verificações têm de ser executadas em hardware. Um TLB poderia servir como cache e melhorar o desempenho da operação de pesquisa.

- 9.6** Um sistema operacional suporta uma memória virtual paginada utilizando um processador central com um ciclo de 1 microsegundo. Tem um custo adicional de 1 microsegundo para acessar uma página que não seja a página corrente. As páginas têm 1000 palavras, e o dispositivo de paginação é um tambor que gira a 3000 rotações por minuto transferindo 1 milhão de palavras por segundo. As seguintes medidas estatísticas foram obtidas do sistema:

- 1% de todas as instruções executadas acessaram uma página diferente da página corrente.

- Das instruções que acessaram outra página, 80% acessaram uma página que já estava em memória.
- Quando uma nova página era necessária, a página substituída tinha sido modificada 50% do tempo.

Calcule o tempo de instrução efetivo nesse sistema, supondo que o sistema esteja executando apenas um processo e que o processador fique ocioso durante as transferências do tambor.

Resposta:

$$\begin{aligned}\text{tempo de acesso efetivo} &= 0,99 \times (1 \mu\text{s} + 0,008 \times (2 \mu\text{s}) \\ &\quad + 0,002 \times (10.000 \mu\text{s} + 1.000 \mu\text{s}) \\ &\quad + 0,001 \times (10.000 \mu\text{s} + 1.000 \mu\text{s})) \\ &= (0,99 \times 0,016 + 22,0 + 11,0) \mu\text{s} \\ &= 34,0 \mu\text{s}\end{aligned}$$

- 9.7 Considere o array bidimensional A:

```
int A[] [] = new int [100] [100] ;
```

em que A[0] [0] está na locação 200 em um sistema de memória paginada com páginas de tamanho 200. Um pequeno processo que manipula a matriz reside na página 0 (locações 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?

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

Resposta:

- 5.000
- 50

- 9.8 Considere a seguinte sequência de referências de páginas:

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 ou sete quadros? Lembre-se de que todos os quadros estão inicialmente vazios, de modo que a primeira página de cada um implicará 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

- 9.9 Suponha que você queira utilizar um algoritmo de paginação que exija um bit de referência (como a substituição da segunda chance ou o modelo do conjunto de trabalho), mas o hardware não forneça um. Esquematize como você poderia simular um bit de referência, mesmo que esse bit não seja fornecido pelo hardware; ou explique por que não é possível fazer isso. Se for possível, calcule qual seria o custo.

Resposta:

Você pode utilizar o bit válido/inválido suportado em hardware para simular o bit de referência. Posicione inicialmente o bit como inválido. Na primeira referência, é gerada uma exceção para o sistema operacional. O sistema operacional posicionará um bit de software como 1 e reposicionará o bit válido/inválido como válido.

- 9.10 Você imaginou um novo algoritmo de substituição de páginas que você 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 pelo tempo mais longo. 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.

- 9.11 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 locações consecutivas suficientes para o segmento requerido. Considere estratégias para sistemas em que os segmentos não possam ser relocados e estratégias para sistemas nos quais 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.

- 9.12 Considere um sistema de computação paginado por demanda em que o grau de multiprogramação esteja cor-

rentemente fixado em quatro. O sistema foi recentemente medido para determinar a utilização da CPU e do disco de paginação. Os resultados são uma das alternativas apresentadas a seguir. O que está ocorrendo 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%; utilização do disco, 97%
- b. Utilização da CPU, 87%; utilização do disco, 3%
- c. Utilização da CPU, 13%; utilização do disco, 3%

Resposta:

- a. Está ocorrendo atividade improdutiva.
- b. A utilização da CPU está suficientemente alta para deixar as coisas como estão e aumentar o grau de multiprogramação.
- c. Aumentar o grau de multiprogramação.

- 9.13 Temos um sistema operacional para uma máquina que utiliza registradores base e limite, mas modificamos a máquina para fornecer uma tabela de páginas. As tabelas de páginas podem ser configuradas para simular registradores base e limite? Como podemos fazer isso, ou por que não podemos fazê-lo?

Resposta:

A tabela de páginas pode ser organizada para simular registradores base e limite, desde que a memória seja aloçada em segmentos de tamanho fixo. Dessa forma, a base de um segmento pode dar entrada na tabela de páginas e o bit válido/inválido utilizado para indicar aquela parte do segmento como residente em memória. Haverá algum problema com fragmentação interna.

Estrutura de Armazenamento de Massa



Exercícios Práticos

- 10.1 O scheduling de disco, exceto o scheduling FCFS, é útil em um ambiente monousuário? Explique sua resposta.

Resposta:

Em um ambiente monousuário, a fila de I/O usualmente está vazia. As solicitações geralmente chegam de um único processo para um bloco ou para uma sequência de blocos consecutivos. Nesses casos, o FCFS é um método econômico para o scheduling de disco. Mas o LOOK é quase tão fácil de programar e oferecerá desempenho muito melhor quando múltiplos processos estiverem executando I/O concorrente, tal como quando um navegador web recupera dados em background enquanto o sistema operacional está paginando e outra aplicação está ativa em foreground.

- 10.2 Explique por que o scheduling SSTF tende a favorecer os cilindros do meio em vez dos cilindros mais internos e mais externos.

Resposta:

O centro do disco é a locação com a menor distância média para todas as outras trilhas. Assim, o cabeçote do disco tende a se mover para longe das margens do disco. Eis aqui outra maneira de pensar sobre isso. A locação corrente do cabeçote divide os cilindros em dois grupos. Se o cabeçote não estiver no centro do disco e chegar uma nova solicitação, essa nova solicitação mais provavelmente estará no grupo que inclui o centro do disco; assim, é mais provável que o cabeçote mover-se à essa direção.

- 10.3 Por que a latência rotacional geralmente não é levada em consideração no scheduling de disco? Como você modificaria o SSTF, o SCAN e o C-SCAN para incluir a otimização da latência?

Resposta:

A maioria dos discos não exporta suas informações de posição rotacional para o hospedeiro. Mesmo que o fizessem, o tempo para que essa informação alcance o scheduler estaria sujeito a imprecisões, e o tempo consumido pelo scheduler é variável; assim, a informação da posição rotacional se tornaria incorreta. Além disso, as solicitações ao disco são usualmente feitas em termos de números de blocos lógicos, e o mapeamento entre blocos lógicos e locações físicas é muito complexo.

- 10.4 Por que é importante equilibrar o I/O do sistema de arquivos entre os discos e os controladores de um sistema em ambiente multitarefa?

Resposta:

Um sistema pode executar somente à velocidade do seu gargalo mais lento. Discos ou controladores de discos representam, frequentemente, o gargalo nos sistemas modernos, já que seus desempenhos individuais não podem sustentar o da CPU e do bus do sistema. Com o平衡amento do I/O entre discos e controladores, nem um disco individual nem um controlador estarão no controle total; portanto, o gargalo será evitado.

- 10.5 Quais são as vantagens e desvantagens envolvidas na releitura de páginas de código a partir do sistema de arquivos *versus* a utilização do espaço de permuta para armazená-las?

Resposta:

Se páginas de código forem armazenadas em espaço de permuta, elas poderão ser transferidas mais rapidamente para a memória principal (porque a alocação do espaço de permuta é ajustada para obter desempenho mais rápido do que a alocação geral do sistema de arquivos). A utilização do espaço de permuta pode requerer tempo de ativação se as páginas forem nele copiadas na invocação do processo, em vez de apenas serem expulsas para o espaço de permuta sob demanda. Além disso, deve ser alocado mais espaço de permuta se ele for utilizado tanto para páginas de código quanto de dados.

- 10.6 Existe alguma maneira de implementar um armazenamento realmente estável? Explique sua resposta.

Resposta:

O armazenamento realmente estável nunca perderia dados. A técnica fundamental para o armazenamento estável é manter múltiplas cópias dos dados de modo que, se uma cópia for destruída, alguma outra cópia ainda estará disponível para uso. Mas, em qualquer esquema, podemos imaginar um desastre suficientemente grande em que todas as cópias sejam destruídas.

- 10.7 Às vezes diz-se que a fita é uma mídia de acesso sequencial, enquanto um disco magnético é uma mídia de acesso aleatório. Na verdade, a adequação de um dispositivo de armazenamento ao acesso aleatório depende do tamanho da transferência. O termo *taxa de escoamento da transferência* denota a taxa para uma transferência de dados que está em curso, excluindo o efeito da latência de acesso. Por outro lado, a *taxa efetiva de transferência* é a relação entre o total de bytes e o total de segundos, incluindo o tempo de overhead por fatores como o tempo de latência.

Suponha que, em um computador, o cache de nível 2 tenha uma latência de acesso de 8 nanossegundos e uma taxa de escoamento da transferência de 800 megabytes por segundo, a memória principal tenha uma latência de acesso de 60 nanossegundos e uma taxa de escoamento da transferência de 80 megabytes por segundo, o disco magnético tenha uma latência de acesso de 15 milissegundos e uma taxa de escoamento da transferência de 5 megabytes por segundo, e um drive de fita tenha uma latência de acesso de 60 segundos e uma taxa de escoamento da transferência de 2 megabytes por segundo.

- O acesso aleatório faz com que a taxa efetiva de transferência de um dispositivo diminua porque não são transferidos quaisquer dados durante o tempo de acesso. Para o disco descrito, qual é a taxa efetiva de transferência se um acesso médio for seguido por uma transferência de escoamento de (1) 512 bytes, (2) 8 quilobytes, (3) 1 megabyte e (4) 16 megabytes?
- O nível de utilização de um dispositivo é a razão entre a taxa efetiva de transferência e a taxa de escoamento da transferência. Calcule o nível de utilização do drive de disco para cada um dos quatro volumes de transferência fornecidos no item a.
- Suponha que um nível de utilização de 25 por cento (ou maior) seja considerado aceitável. Usando os números de desempenho fornecidos, calcule o menor tamanho de transferência de disco que ofereça um nível de utilização aceitável.
- Complete a frase a seguir: um disco é um dispositivo de acesso aleatório para transferências maiores do que _____ bytes e um dispositivo de acesso sequencial para transferências menores.
- Calcule os tamanhos mínimos de transferência que ofereçam um nível de utilização aceitável para o cache, a memória e a fita.
- Quando uma fita é um dispositivo de acesso aleatório e quando é um dispositivo de acesso sequencial?

Resposta:

- Para 512 bytes, a taxa efetiva de transferência (TET) é calculada da forma a seguir.
 $TET = \text{volume de transferência} / \text{tempo de transferência}$.
 Se X é o volume de transferência, então o tempo de transferência será de $((X / TDET) (\text{taxa de escoamento da transferência})) + \text{latência}$.
 O tempo de transferência é de $15\text{ms} + (512\text{B}/5\text{MB por segundo}) = 15,0097\text{ms}$.
 A taxa efetiva de transferência é, portanto,
 $512\text{B}/15,0097\text{ms} = 33,12\text{ KB/s}$.
 TET para 8 KB = 0,47 MB/s.
 TET para 1 MB = 4,65 MB/s.
 TET para 16 MB = 4,98 MB/s.

- Utilização do dispositivo para 512 B = $33,12\text{ KB/s} / 5\text{ MB/s} = 0,0064 = 0,64$.

Para 8 KB = 9,4%.

Para 1 MB = 93%.

Para 16 MB = 99,6%.

- Calcule $0,25 = TET/TDET$, para o volume de transferência X.

$TDET = 5\text{MB}$; assim, $1,25\text{MB/s} = TET$.

$$1,25\text{MB/s} * ((X/5) + 0,015) = X.$$

$$0,25X + 0,01875 = X.$$

$$X = 0,025\text{MB}.$$

- Um disco é um dispositivo de acesso aleatório para transferências maiores do que K bytes (em que $K >$ tamanho do bloco do disco), e é um dispositivo de acesso sequencial para transferências menores.

- Calcule o volume de transferência mínimo para a utilização aceitável da memória do cache.

$TDET = 800\text{ MB}$, $TET = 200$, latência = $8 * 10^{-9}$.

$$200(X\text{MB}/800 + 8 * 10^{-9}) = X\text{MB}.$$

$$0,25X\text{MB} + 1600 * 10^{-9} = X\text{MB}.$$

$$X = 2,24\text{ bytes}.$$

Calcule para a memória:

$TET = 80\text{ MB}$, $TTE = 20$, $L = 60 * 10^{-9}$.

$$20(X\text{MB}/80 + 60 * 10^{-9}) = X\text{MB}.$$

$$0,25X\text{MB} + 1200 * 10^{-9} = X\text{MB}.$$

$$X = 1,68\text{ bytes}.$$

Calcule para fita:

$TDET = 2\text{ MB}$, $TET = 0,5$, $L = 60\text{s}$.

$$0,5(X\text{MB}/2 + 60) = X\text{MB}.$$

$$0,25X\text{MB} + 30 = X\text{MB}.$$

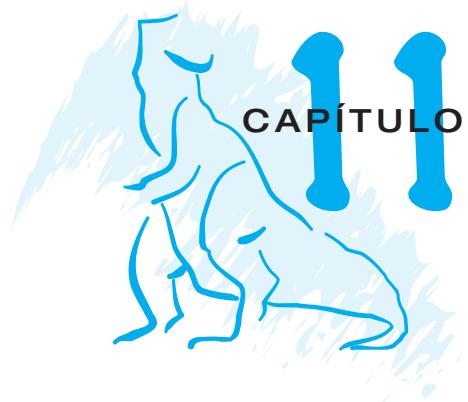
$$X = 40\text{MB}.$$

- Depende de como ela está sendo utilizada. Suponha que estamos utilizando a fita para restaurar um backup. Nessa instância, uma fita atua como um dispositivo de acesso sequencial em que estamos lendo sequencialmente o conteúdo da fita. Como outro exemplo, suponha que estamos utilizando a fita para acessar uma variedade de registros nela armazenados. Nessa instância, o acesso à fita é arbitrário e, portanto, considerado aleatório.

- Uma organização RAID nível 1 poderia obter um desempenho melhor para solicitações de leitura do que uma organização RAID nível 0 (com distribuição de dados sem redundância)? Em caso afirmativo, como?

Resposta:

Sim, uma organização RAID nível 1 poderia obter um desempenho melhor para solicitações de leitura. Quando uma operação de leitura é executada, um sistema RAID nível 1 pode decidir qual das duas cópias do bloco deve ser acessada para atender a solicitação. Essa opção poderia ser baseada na locação corrente do cabeçote do disco e, portanto, resultar em otimizações no desempenho pela seleção de um cabeçote mais próximo dos dados desejados.



Interface do Sistema de Arquivos

Exercícios Práticos

- 11.1 Alguns sistemas apagam automaticamente todos os arquivos de usuário quando o usuário se desconecta ou um job termina, a menos que o usuário solicite explicitamente que eles sejam mantidos; outros sistemas mantêm todos os arquivos, a não ser que o usuário os apague explicitamente. Discuta os méritos relativos de cada abordagem.

Resposta:

O apagamento de todos os arquivos não salvos especificamente pelo usuário tem a vantagem de minimizar o espaço de arquivo necessário para cada usuário pelo não salvamento de arquivos indesejados ou desnecessários. O salvamento de todos os arquivos, a menos que sejam especificamente apagados, é mais seguro para o usuário no sentido de que não será possível perder arquivos inadvertidamente pelo esquecimento de salvá-los.

- 11.2 Por que alguns sistemas controlam o tipo de um arquivo enquanto outros delegam essa função ao usuário, e outros simplesmente não implementam múltiplos tipos de arquivo? Qual sistema é “melhor”?

Resposta:

Alguns sistemas permitem diferentes operações de arquivo baseadas no tipo do arquivo (por exemplo, um arquivo ascii pode ser lido como uma cadeia, enquanto um arquivo de banco de dados pode ser lido por meio de um índice para um bloco). Outros sistemas deixam tal interpretação dos dados de um arquivo para o processo e não fornecem ajuda no acesso aos dados. O método que é “melhor” depende das necessidades dos processos no sistema e das demandas que os usuários impõem ao sistema operacional. Se um sistema executa principalmente aplicações de bancos de dados, pode ser mais eficiente para o sistema operacional implementar um arquivo do tipo banco de dados e fornecer operações, em vez de fazer cada programa implementar a mesma coisa (possivelmente de diferentes maneiras). Para sistemas de uso geral, pode ser melhor implementar somente tipos básicos de arquivos para manter o tamanho do sistema operacional menor e permitir o máximo de liberdade aos processos no sistema.

- 11.3 De modo semelhante, 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 serem 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.

- 11.4 Você poderia simular uma estrutura de diretórios multinível usando uma estrutura de diretórios em um único nível na qual nomes arbitrariamente longos possam ser utilizados? Se sua resposta for sim, explique como pode fazer isso e compare esse esquema com o esquema do diretório multinível. Se sua resposta for não, explique o que impede o sucesso da sua simulação. Como sua resposta mudaria se os nomes de arquivos fossem limitados a sete caracteres?

Resposta:

Se nomes arbitrariamente longos puderem ser utilizados, então será possível simular uma estrutura de diretórios multinível. Isso pode ser feito, por exemplo, utilizando-se o caractere “.” para indicar o fim de um subdiretório. Assim, por exemplo, o nome *jim.java.A1* especifica que *A1* é um arquivo no subdiretório *java* que, por sua vez, está no diretório raiz *jim*.

Se os nomes de arquivo forem limitados a sete caracteres, então o esquema acima não poderia ser utilizado e, assim, em geral, a resposta é *não*. A próxima melhor abordagem nessa situação seria utilizar um arquivo específico

como tabela de símbolos (diretório) para mapear nomes arbitrariamente longos (tais como *jim.java.A1*) em nomes arbitrários mais curtos (tais como *XX00743*), que são, então, utilizados para acesso real ao arquivo.

- 11.5** 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 de fechamento.

- 11.6** Em alguns sistemas, um subdiretório pode ser lido e gravado por um usuário autorizado, do mesmo modo que os arquivos comuns.

- a. Descreva os problemas de proteção que podem surgir.
- b. Sugira um esquema para lidar com cada um desses problemas de proteção.

Resposta:

- a. Uma parte da informação mantida em uma entrada de diretório é a locação do arquivo. Se um usuário puder modificar essa locação, então ele poderá acessar outros arquivos derrotando o esquema de proteção de acesso.
- b. Não permitir que o usuário grave diretamente no subdiretório. Em vez disso, oferecer operações do sistema para fazer isso.

- 11.7** Considere um sistema que suporte 5.000 usuários. Suponha que você queira permitir acesso a um arquivo a 4.990 desses usuários.

- a. Como você especificaria esse esquema de proteção no UNIX?

- b. Você poderia sugerir outro esquema de proteção que possa ser utilizado de modo mais eficaz para esse propósito do que o esquema fornecido pelo UNIX?

Resposta:

- a. Existem dois métodos para alcançar isso:
 - i. Criar uma lista de controle de acesso com os nomes de todos os 4.990 usuários.
 - ii. Colocar esses 4.990 usuários em um grupo e definir o acesso ao grupo de forma conveniente. Esse esquema nem sempre pode ser implementado, já que grupos de usuários são restritos pelo sistema.
- b. O acesso universal a arquivos aplica-se a todos os usuários, a menos que seus nomes apareçam na lista de controle de acesso com diferentes permissões de acesso. Com esse esquema, você simplesmente coloca os nomes dos 10 usuários remanescentes na lista de controle de acesso, mas sem acessos privilegiados permitidos.

- 11.8** Pesquisadores têm sugerido que, em vez de termos uma lista de acesso associada a cada arquivo (especificando quais usuários podem acessar o arquivo e como), poderíamos ter uma *lista de controle de usuários* associada a cada usuário (especificando quais arquivos um usuário pode acessar e como). Discuta os méritos relativos desses dois esquemas.

Resposta:

- *Lista de controle de acesso.* Como as informações de controle de acesso são concentradas em um único local, é mais fácil alterá-las, e isso requer menos espaço.
- *Lista de controle de usuários.* Esse recurso impõe menos overhead ao abrir um arquivo.



Implementação do Sistema de Arquivos

Exercícios Práticos

- 12.1** Considere um arquivo contendo correntemente 100 blocos. Suponha que o bloco de controle do arquivo (e o bloco de índices, no caso da alocação indexada) já esteja em memória. Calcule quantas operações de I/O de disco são necessárias para as estratégias de alocação contígua, encadeada e indexada (em nível único) se as condições a seguir forem mantidas para um bloco. No caso da alocação contígua, considere que não exista espaço para crescimento no início, mas exista espaço no final. Considere também que as informações do bloco a ser adicionado estão armazenadas em memória.
- O bloco é adicionado no início.
 - O bloco é adicionado no meio.
 - O bloco é adicionado no final.
 - O bloco é removido do início.
 - O bloco é removido do meio.
 - O bloco é removido do final.

Resposta:

Os resultados são:

	Contígua	Encadeada	Indexada
a.	201	1	1
b.	101	52	1
c.	1	3	1
d.	198	1	0
e.	98	52	0
f.	0	100	0

- 12.2** Que problemas poderiam ocorrer se um sistema permitisse que um sistema de arquivos fosse montado simultaneamente em mais de uma locação?

Resposta:

Existiriam múltiplos caminhos para o mesmo arquivo, o que poderia confundir os usuários ou encorajar erros (a exclusão de um arquivo com um caminho exclui o arquivo em todos os outros caminhos).

- 12.3** 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 (falla 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.

- 12.4** 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.

- 12.5** 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 esse problema é definir uma estrutura de arquivo consistindo em 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 essa implementação de um arquivo com as implementações contígua padrão e encadeada padrão.

Resposta:

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

- 12.6** 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.

- 12.7 Por que é vantajoso para o usuário que um sistema operacional aloque dinamicamente suas tabelas internas? Que desvantagens isso traz para o sistema operacional?

Resposta:

Tabelas dinâmicas permitem maior flexibilidade no crescimento de uso do sistema — tabelas nunca são excedidas, evitando limites artificiais ao uso. Infelizmente, as estruturas e o código do kernel são mais complicados; portanto, existe maior potencial para bugs. O uso de um recurso pode comprometer mais recursos do sistema (pe-
lo crescimento, para acomodar as solicitações) do que com as tabelas estáticas.

- 12.8 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 ações indiretas 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.



Sistemas de I/O

Exercícios Práticos

- 13.1 Cite três vantagens de alocar funcionalidade a um controlador de dispositivo em vez de alocá-la ao kernel. Cite três desvantagens.

Resposta:

Três vantagens:

- Há menos probabilidade de os bugs causarem uma queda do sistema operacional.
- O desempenho pode ser aumentado pela utilização de hardware dedicado e algoritmos incluídos no código fonte.
- O kernel é simplificado pela movimentação dos algoritmos para fora dele.

Três desvantagens:

- Os bugs são mais difíceis de corrigir — é necessária uma nova versão firmware ou um novo hardware.
- A melhoria dos algoritmos requer do mesmo modo uma atualização do hardware, em vez de apenas uma atualização do kernel ou do driver de dispositivo.
- Algoritmos embutidos podem conflitar com o uso do dispositivo da aplicação, causando degradação de desempenho.

- 13.2 O exemplo de aperto de mãos da Seção 13.2 utilizou 2 bits: um bit ocupado e um bit de comando pronto. É possível implementar esse aperto de mãos com apenas um bit? Se for possível, descreva o protocolo. Se não for, explique por que 1 bit é insuficiente.

Resposta:

É possível, utilizando o algoritmo a seguir. Vamos supor que simplesmente utilizemos o bit ocupado (ou o bit de comando pronto; essa resposta é a mesma, de qualquer maneira). Quando o bit está desligado, o controlador está ocioso. O hospedeiro grava dados de saída e posiciona o bit para indicar que uma operação está pronta (o equivalente ao posicionamento do bit de comando pronto). Quando o controlador tiver terminado, ele desliga o bit ocupado. O hospedeiro inicia, então, a próxima operação. Essa solução requer que tanto o hospedeiro quanto o controlador tenham acesso de leitura e gravação para o mesmo bit, o que pode complicar o sistema de circuitos e aumentar o custo do controlador.

- 13.3 Por que um sistema pode utilizar I/O dirigido por interrupções para gerenciar uma porta serial única e o I/O

com inquirição para gerenciar um processador front-end, tal como um concentrador de terminais?

Resposta:

O I/O com inquirição pode ser mais eficiente do que o I/O dirigido por interrupções. Esse é o caso, quando o I/O é frequente e de curta duração. Mesmo que uma única porta serial execute I/O com relativa infreqüência, podendo, assim, utilizar interrupções, uma coleção de portas seriais, como as de um concentrador de terminais, pode produzir muitas operações de I/O curtas, e a interrupção de cada uma dessas operações poderia gerar uma carga pesada sobre o sistema. Um bem cadenciado loop de inquirições poderia aliviar essa carga sem desperdiçar muitos recursos por meio da execução do loop sem necessidade de I/O.

- 13.4 A inquirição para conclusão de um I/O pode desperdiçar um grande número de ciclos de CPU se o processador realizar muitos loops de espera em ação antes que o I/O se complete. Mas, se o dispositivo de I/O estiver pronto para o serviço, a inquirição poderá ser muito mais eficiente do que a captura e o despacho de uma interrupção. Descreva uma estratégia híbrida que combine inquirição, suspensão e interrupções para o serviço dos dispositivos de I/O. Para cada uma dessas três estratégias (inquirição pura, interrupções puras e um híbrido), descreva um ambiente computacional no qual a estratégia em pauta seja mais eficiente do que as outras duas.

Resposta:

Uma abordagem híbrida pode comutar entre inquirição e interrupções, dependendo do tamanho da espera da operação de I/O. Por exemplo, poderíamos inquirir e executar o loop N vezes, e, se o dispositivo ainda estiver ocupado em N + 1, poderíamos posicionar uma interrupção e dormir. Essa abordagem evitaria longos ciclos de espera em ação. Esse método seria melhor para tempos de ocupação muito longos ou muito curtos. O método seria ineficiente se o I/O se completar em N + T (em que T é um número pequeno de ciclos) por causa do overhead da inquirição mais o posicionamento e a captura das interrupções.

A inquirição pura é melhor com tempos de espera muito curtos. As interrupções são melhores com longos tempos de espera conhecidos.

- 13.5 Como o DMA aumenta a concorrência no sistema? Como ele complica o projeto de hardware?

Resposta:

O DMA aumenta a concorrência no sistema ao permitir que a CPU execute tarefas enquanto o sistema DMA transfere dados por meio dos buses do sistema e da memória. O projeto do hardware é complicado porque o controlador DMA deve ser integrado ao sistema, e o sistema deve permitir que o controlador DMA seja um bus mestre. O roubo de ciclos também pode ser necessário para permitir que a CPU e o controlador DMA compartilhem o uso do bus da memória.

- 13.6 Por que é importante aumentar as velocidades do bus do sistema e dos dispositivos, à medida que a velocidade da CPU aumenta?

Resposta:

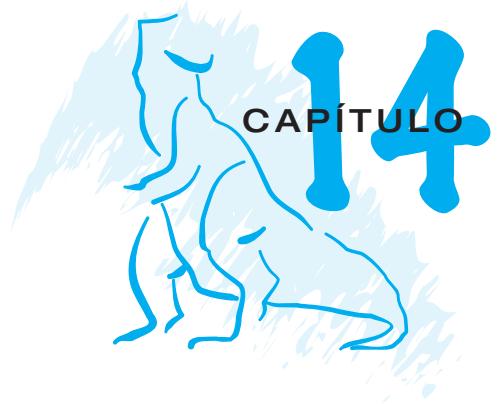
Considere um sistema que execute 50% de I/O e 50% de computação. A duplicação do desempenho da CPU nes-

se sistema aumentaria o desempenho total do sistema em apenas 50%. A duplicação de ambos os aspectos do sistema aumentaria o desempenho em 100%. Geralmente, é importante remover o gargalo corrente no sistema e aumentar o desempenho global do sistema, em vez de aumentar cegamente o desempenho dos componentes individuais do sistema.

- 13.7 Diferencie um driver do STREAMS de um módulo do STREAMS.

Resposta:

O driver do STREAMS controla um dispositivo físico que pode estar envolvido em uma operação do STREAMS. O módulo do STREAMS modifica o fluxo de dados entre a cabeça (a interface do usuário) e o driver.



Proteção

Exercícios Práticos

- 14.1 Quais são as principais diferenças entre listas de competências e listas de acessos?

Resposta:

Uma lista de acessos é uma lista para cada objeto que consiste em domínios, com um conjunto não vazio de direitos de acesso para o objeto. Uma lista de competências é uma lista de objetos e as operações permitidas sobre esses objetos para cada domínio.

- 14.2 Um arquivo MCP no Burroughs B7000/B6000 pode ser etiquetado como dados sensíveis. Quando tal arquivo é apagado, sua área de armazenamento é sobreposta por alguns bits aleatórios. Para que propósito tal esquema seria útil?

Resposta:

Isso poderia ser útil como medida de segurança extra, de modo que o conteúdo antigo da memória não possa ser acessado por outro programa, tanto intencionalmente como por acidente. Isso é especialmente útil para qualquer informação altamente confidencial.

- 14.3 Em um sistema de proteção em anel, o nível 0 possui o maior nível de acesso aos objetos, e o nível n (em que $n > 0$) possui direitos de acesso menores. Os direitos de acesso de um programa em determinado nível na estrutura de anel são considerados um conjunto de competências. Qual é a relação entre as competências de um domínio no nível j e de um domínio no nível i para um objeto (para $j > i$)?

Resposta:

D_j é um subconjunto de D_i .

- 14.4 O sistema RC 4000, entre outros, definiu uma hierarquia de processos (chamada árvore de processos) de modo tal que todos os descendentes de um processo podem receber recursos (objetos) e direitos de acesso somente de seus ancestrais. Assim, um descendente não poderá jamais fazer algo que seus ancestrais não possam fazer. A raiz da árvore é o sistema operacional, que pode fazer qualquer coisa. Suponha que o conjunto de direitos de acesso seja representado por uma matriz de acesso, A . $A(x,y)$ define os direitos de acesso do processo x para o objeto y . Se x for um descendente de z , qual será o relacionamento entre $A(x,y)$ e $A(z,y)$ para um objeto arbitrário y ?

Resposta:

$A(x,y)$ é um subconjunto de $A(z,y)$.

- 14.5 Que problemas de proteção podem surgir se uma pilha compartilhada for utilizada para passagem de parâmetros?

Resposta:

O conteúdo da pilha poderia ser comprometido por outro(s) processo(s) que estivesse(m) compartilhando a pilha.

- 14.6 Considere um ambiente computacional em que um único número seja associado a cada processo e a cada objeto no sistema. Suponha que um processo com o número n seja autorizado a acessar um objeto com o número m somente no caso de $n > m$. Que tipo de estrutura de proteção teremos?

Resposta:

Estrutura hierárquica.

- 14.7 Considere um ambiente computacional em que é concedido a um processo o privilégio de acessar um objeto somente n vezes. Sugira um esquema para a implementação dessa política.

Resposta:

Adicione um contador inteiro à competência.

- 14.8 Se todos os direitos de acesso de um objeto forem excluídos, o objeto não poderá mais ser acessado. Nesse caso, o objeto também deveria ser excluído e o espaço que ele ocupa deveria ser retornado ao sistema. Sugira uma implementação eficiente desse esquema.

Resposta:

Contagens de referências.

- 14.9 Por que é difícil proteger um sistema no qual os usuários são autorizados a fazer seu próprio I/O?

Resposta:

Em capítulos anteriores, identificamos uma diferença entre modalidade de kernel e de usuário em que a modalidade de kernel é utilizada para realizar operações privilegiadas, tais como I/O. Uma razão para que I/O deva ser executado em modalidade de kernel é que o I/O requer acesso ao hardware, e o acesso apropriado ao hardware é necessário para a integridade do sistema. Se permitirmos que os usuários executem o seu próprio I/O, não poderemos garantir a integridade do sistema.

- 14.10 As listas de competências normalmente são mantidas dentro do espaço de endereçamento do usuário. Como o sistema garante que o usuário não pode modificar o conteúdo da lista?

Resposta:

Uma lista de competências é considerada um “objeto protegido” e é acessada apenas indiretamente pelo usuário. O sistema operacional garante que o usuário não possa acessar a lista de competências diretamente.



Segurança

Não existem Exercícios Práticos.



Máquinas Virtuais

Não existem Exercícios Práticos.



Sistemas Distribuídos

Exercícios Práticos

- 17.1 Por que seria uma má ideia que os gateways passassem pacotes de disseminação entre redes? Que vantagens isso traria?

Resposta:

Todas as difusões poderiam ser propagadas para todas as redes, causando *muito* tráfego na rede. Se o tráfego de difusão for limitado aos dados importantes (e a muito pouco destes), então a propagação da difusão dispensaria os gateways de terem que executar software especial para esperar por esses dados (tais como informações de roteamento da rede) e difundi-los novamente.

- 17.2 Discuta as vantagens e desvantagens de armazenar em cache traduções de nomes para computadores localizados em domínios remotos.

Resposta:

Existe uma vantagem de desempenho no armazenamento em cache das traduções de nomes para computadores localizados em domínios remotos: a resolução repetida do mesmo nome, a partir de diferentes computadores localizados no domínio local, pode ser executada localmente sem requerer uma operação remota de busca do nome. A desvantagem é que poderiam existir inconsistências nas traduções de nomes quando forem feitas atualizações no mapeamento de nomes em endereços IP. Esses problemas de consistência podem ser resolvidos invalidando as traduções, o que exigiria que o estado fosse gerenciado sem levar em conta quais computadores estariam armazenando em cache determinada tradução e, também, demandaria várias mensagens de invalidação. Também poderiam ser utilizados prazos de arrendamento, por meio dos quais a entidade armazenada em cache invalidaria uma tradução após determinado período de tempo. A última abordagem requer menos estados e não requer mensagens de invalidação, mas pode sofrer de inconsistências temporárias.

- 17.3 Quais são as vantagens e desvantagens do uso da comutação por circuitos? Para que tipos de aplicações a comutação por circuitos é uma estratégia viável?

Resposta:

A comutação por circuitos garante que os recursos da rede requeridos para uma transferência sejam reservados antes que a transmissão tenha lugar. Isso garante que os pacotes não seriam abandonados e sua transmissão satisfaria os requisitos de qualidade de serviço. A desvantagem

da comutação por circuitos é que ela requer uma mensagem de ida e volta para posicionar as reservas e também pode superprovisionar recursos, resultando na subutilização desses recursos. A comutação por circuitos é uma estratégia viável para aplicações que tenham demandas constantes, sem levar em conta os recursos da rede, e que precisariam dos recursos por longo período de tempo, amortizando, dessa forma, os overheads iniciais.

- 17.4 Quais são dois grandes problemas que os projetistas devem resolver para implementar um sistema de rede transparente?

Resposta:

Um desses problemas é fazer todos os processadores e dispositivos de armazenamento parecerem transparentes através da rede. Em outras palavras, o sistema distribuído deve ter a aparência de um sistema centralizado para os usuários. O sistema de arquivos Andrew e o NFS fornecem esse recurso: o sistema de arquivos distribuído aparece para os usuários como um único sistema de arquivos, mas, na realidade, ele pode ser distribuído por uma rede.

Outro problema refere-se à mobilidade dos usuários. Queremos permitir que os usuários se conectem ao "sistema" e não a uma máquina específica (embora, na verdade, eles possam estar se conectando a uma máquina específica em algum lugar do sistema distribuído).

- 17.5 A migração de processos dentro de uma rede heterogênea é normalmente impossível, dadas as diferenças das arquiteturas e dos sistemas operacionais. Descreva um método para a migração de processos por meio de diferentes arquiteturas executando:

- O mesmo sistema operacional
- Sistemas operacionais diferentes

Resposta:

No mesmo sistema operacional, a migração de processos é relativamente direta, já que o estado do processo precisa migrar de um processador para outro. Isso envolve a movimentação do espaço de endereçamento, do estado dos registradores da CPU, e dos arquivos abertos do sistema fonte para o sistema de destino. Entretanto, é importante que cópias idênticas do sistema operacional estejam em execução nos sistemas diferentes para garantir a compatibilidade. Se os sistemas operacionais forem os mesmos, mas talvez estejam em execução versões diferentes nos sistemas separados, então a migração de pro-

cessos deve estar segura de seguir roteiros de programação que sejam consistentes entre as diferentes versões do sistema operacional.

As applets Java fornecem um bom exemplo de migração de processos entre sistemas operacionais diferentes. Para ocultar diferenças no sistema subjacente, o processo migrado (isto é, a applet Java) executa em máquina virtual em vez de em um sistema operacional específico. Tudo que é requerido é que a máquina virtual esteja em execução no sistema para o qual o processo migrar.

- 17.6** Para construir um sistema distribuído robusto, você deve saber quais tipos de falhas podem ocorrer.
- Liste três tipos de falha possíveis em um sistema distribuído.
 - Especifique quais das entradas de sua lista também são aplicáveis a um sistema centralizado.

Resposta:

Três falhas comuns em um sistema distribuído incluem: (1) falha do link de rede, (2) falha do hospedeiro, e (3) falha do meio de armazenamento. Tanto (2) quanto (3) são falhas que também poderiam ocorrer em um sistema centralizado, enquanto uma falha do link de rede somente pode ocorrer em um sistema distribuído em rede.

- 17.7** É sempre crucial saber se a mensagem que você enviou chegou em seu destino com segurança? Se sua resposta for *sim*, explique por quê. Se sua resposta for *não*, forneça exemplos apropriados.

Resposta:

Não. Muitos programas coletores de estados funcionam a partir da suposição de que os pacotes não podem ser recebidos pelo sistema de destino. Esses programas geralmente *distribuem* um pacote e assumem que pelo menos alguns outros sistemas em sua rede receberão a informação. Por exemplo, um daemon em cada sistema pode distribuir a carga média e o número de usuários do

sistema. Essas informações podem ser utilizadas para a seleção do alvo na migração de processos. Outro exemplo é um programa que determina se um sítio remoto está em execução e é acessível na rede. Se o programa enviar uma consulta e não obtiver resposta, ele saberá que o sistema não pode ser, correntemente, alcançado.

- 17.8** Um sistema distribuído tem dois sítios, A e B. Avalie se o sítio A pode distinguir entre os eventos a seguir:
- B fica inativo.
 - O link entre A e B fica inativo.
 - B está extremamente sobrecarregado, e seu tempo de resposta é 100 vezes mais longo do que o normal.

Que implicações sua resposta tem na recuperação em sistemas distribuídos?

Resposta:

Uma técnica seria que B envie periodicamente uma mensagem *Eu-estou-ativo* para A, indicando que ainda está vivo. Se A não receber uma mensagem *Eu-estou-ativo*, pode supor que B — ou o link de rede — caiu. Observe que uma mensagem *Eu-estou-ativo* não permite que A diferencie cada tipo de falha. Uma técnica que permite a A determinar melhor se a rede caiu é enviar uma mensagem *Você-está-ativo* a B utilizando uma rota alternativa. Se receber uma resposta, ele pode determinar que realmente o link de rede caiu e que B está ativo.

Se assumirmos que A sabe que B está ativo e é alcançável (por meio do mecanismo *Eu-estou-ativo*) e que A tem algum valor *N* que indique um tempo de resposta normal, A poderia monitorar o tempo de resposta de B e comparar valores a *N*, permitindo que A determine se B está sobrecarregado ou não.

As implicações dessas duas técnicas são que A poderia escolher outro hospedeiro — digamos C — no sistema, se B estiver caído, for inalcançável ou estiver sobrecarregado.



O Sistema Linux

Exercícios Práticos

- 18.1 Módulos do kernel dinamicamente carregáveis oferecem flexibilidade quando drivers são adicionados a um sistema, mas eles também apresentam desvantagens? Sob que circunstâncias um kernel seria compilado em um único arquivo binário, e quando seria melhor mantê-lo dividido em módulos? Explique sua resposta.

Resposta:

Existem duas desvantagens principais no uso de módulos. A primeira é o tamanho: o gerenciamento de módulos consome memória não paginável do kernel, e um kernel básico, com vários módulos carregados, consumirá mais memória do que um kernel equivalente com os drivers compilados na própria imagem do kernel. Esse pode ser um aspecto muito significativo em máquinas com memória física limitada.

A segunda desvantagem é que módulos podem aumentar a complexidade do processo de bootstrap do kernel. É difícil carregar um conjunto de módulos do disco se o driver precisar, ele mesmo, acessar, no disco, um módulo a ser carregado. Como resultado, o gerenciamento do bootstrap do kernel com módulos pode exigir trabalho extra por parte do administrador: os módulos requeridos para bootstrap precisarão ser colocados em uma imagem de disco RAM que seja carregada em paralelo com a imagem inicial do kernel quando o sistema é inicializado.

Em certos casos, é melhor utilizar um kernel modular; em outros casos, é melhor utilizar um kernel com seus drivers de dispositivos pré-vinculados. Onde for importante a minimização do tamanho do kernel, a escolha dependerá da frequência de uso dos vários drivers de dispositivos. Se eles forem utilizados constantemente, então os módulos serão inadequados. Isso é especialmente verdadeiro onde os drivers são necessários para o próprio processo de inicialização. Por outro lado, se alguns drivers não forem sempre necessários, então o mecanismo de módulos permitirá que esses drivers sejam carregados e descarregados sob demanda, oferecendo potencialmente uma economia final da memória física.

Em casos em que um kernel deve ser construído para uso em uma ampla variedade de máquinas muito diferentes, então a sua construção com módulos será claramente preferível ao uso de um kernel único com dúzias de drivers desnecessários consumindo memória. Esse é particularmente o caso de kernels distribuídos comercial-

mente, em que o suporte à mais ampla variedade de hardwares da maneira mais simples possível é uma prioridade.

Entretanto, se um kernel está sendo construído para uma única máquina cuja configuração é conhecida com antecedência, então a compilação e o uso de módulos poderão ser simplesmente uma complexidade desnecessária. Em casos como esse, o uso de módulos poderá bem ser uma questão de gosto.

- 18.2 A criação de múltiplos threads é uma técnica de programação comumente utilizada. Descreva três maneiras diferentes de implementar threads, e compare esses três métodos com o mecanismo clone() do Linux. Quando a utilização de cada mecanismo alternativo pode ser melhor ou pior do que o uso de clones?

Resposta:

As implementações de threads podem ser, de modo geral, classificadas em dois grupos: threads baseados no kernel e threads de modalidade de usuário. Os pacotes de threads de modalidade de usuário apoiam-se em algum suporte do kernel — eles podem requerer recursos de interrupção por timer, por exemplo —, mas o scheduling entre os threads não é executado pelo kernel e, sim, por alguma biblioteca de código em modalidade de usuário. Múltiplos threads em tal implementação aparecem para o sistema operacional como um único contexto de execução. Quando um processo multithreads está em execução, ele decide por si mesmo quais de seus threads devem ser executados, utilizando saltos não locais para realizar a comutação entre os threads de acordo com suas próprias regras de scheduling preemptivo ou não preemptivo.

Alternativamente, o kernel do sistema operacional pode fornecer, ele mesmo, suporte para threads. Nesse caso, os threads podem ser implementados como processos separados que compartilham um espaço de endereçamento comum, completo ou parcial, ou eles podem ser implementados como contextos de execução separados dentro de um único processo. Seja qual for a forma pela qual os threads são organizados, eles aparecem como contextos de execução completamente independentes para a aplicação.

Também são possíveis implementações híbridas, em que um grande número de threads se torna disponível para a aplicação, utilizando um número menor de threads do kernel. Threads executáveis de usuário são executados pelo primeiro thread de kernel disponível.

No Linux, os threads são implementados dentro do kernel por um mecanismo de clone que cria um novo processo dentro do mesmo espaço de endereçamento virtual do processo-pai. Diferentemente de alguns pacotes de threads baseados no kernel, o kernel do Linux não faz nenhuma distinção entre threads e processos: um thread é simplesmente um processo que não criou um novo espaço de endereçamento virtual quando foi inicializado.

A principal vantagem da implementação de threads no kernel em vez de em uma biblioteca de modalidade de usuário é que:

- sistemas com threads no kernel podem tirar vantagem de múltiplos processadores, se eles estiverem disponíveis; e
- se um thread bloquear em uma rotina de serviço do kernel (por exemplo, uma chamada de sistema ou um erro de página), outros threads ainda serão capazes de executar.

Uma vantagem menor é a capacidade de designar diferentes atributos de segurança a cada thread.

As implementações em modalidade de usuário não apresentam essas vantagens. Como tais implementações executam inteiramente dentro de um único contexto de execução do kernel, apenas um thread pode estar em execução de cada vez, mesmo se estão disponíveis múltiplas CPUs. Pela mesma razão, se um thread entra em uma chamada de sistema, nenhum outro thread pode ser executado até que a chamada de sistema se complete. Como resultado, um thread, ao fazer uma leitura de disco com bloqueio, suspenderá todos os threads da aplicação. Entretanto, as implementações em modalidade de usuário têm suas próprias vantagens. A mais óbvia é o desempenho: a invocação do próprio scheduler do kernel para realizar a comutação entre threads envolve a entrada em um novo domínio de proteção enquanto a CPU comuta para a modalidade de kernel, ao passo que a comutação entre threads em modalidade de usuário pode ser alcançada simplesmente salvando e restaurando os registradores principais da CPU. Os threads em modalidade de usuário também podem consumir menos memória do sistema: a maior parte dos sistemas UNIX reservará pelo menos uma página completa para uma pilha do kernel para cada thread do kernel, e essa pilha pode não ser paginável.

A abordagem híbrida, implementando múltiplos threads de usuário em contrapartida a um número menor de threads do kernel, permite alcançar um equilíbrio entre essas decisões. Os threads do kernel permitirão que múltiplos threads estejam em chamadas do kernel com bloqueio de uma vez, e permitirão a execução em múltiplas CPUs; além disso, a comutação entre threads de modalidade de usuário poderá ocorrer dentro de cada thread do kernel para executar um threading peso leve, sem os overheads da presença de muitos threads de kernel. O lado negativo dessa abordagem é a complexidade: o controle sobre as decisões complica a interface de usuário da biblioteca de threads.

18.3 O kernel do Linux não permite a paginação para fora da memória do kernel. Que efeito essa restrição tem sobre o projeto do kernel? Quais são duas vantagens e duas desvantagens dessa decisão de projeto?

Resposta:

O principal impacto da não permissão de paginação da memória do kernel no Linux é que a não possibilidade de

preempção do kernel é preservada. Qualquer processo que capture um erro de página, seja em modalidade de kernel ou de usuário, corre o risco de ser realocado no schedule enquanto os dados requeridos estão sendo paginados do disco para a memória. Como o kernel pode se valer do fato de não ser realocado no schedule durante o acesso às suas estruturas de dados principais, os requisitos de trancamento para proteger a integridade dessas estruturas de dados são muitíssimo simplificados. Embora a simplicidade de projeto seja um benefício em si mesma, ela também oferece uma importante vantagem de desempenho em máquinas uniprocessadoras pelo fato de não ser necessário fazer trancamento adicional na maioria das estruturas de dados internas.

Existem, porém, muitas desvantagens na falta de memória paginável no kernel. Em primeiro lugar, ela impõe restrições ao montante de memória que o kernel pode utilizar. Não é razoável manter estruturas de dados muito grandes em memória não paginável, já que isso representa memória física que absolutamente não pode ser utilizada por mais ninguém. Isso tem dois impactos: primeiro, o kernel deve remover muitas de suas estruturas de dados internas manualmente, em vez de poder se valer de um único mecanismo de memória virtual para manter o uso da memória física sob controle; segundo, isso torna inviável implementar determinados recursos que exigem grandes montantes de memória virtual no kernel, tais como o sistema de arquivos /tmp (um sistema de arquivos rápido baseado em memória virtual encontrado em alguns sistemas UNIX).

Observe que a complexidade de gerenciamento dos erros de página enquanto está em execução o código do kernel não é um problema aqui. O código do kernel do Linux já é capaz de lidar com erros de página: ele precisa ser capaz de lidar com chamadas de sistema cujos argumentos referenciam a memória do usuário que pode ser paginada da memória para disco.

18.4 Discuta três vantagens da vinculação dinâmica (compartilhada) de bibliotecas em comparação com a vinculação estática. Descreva dois casos em que a vinculação estática é preferível.

Resposta:

As principais vantagens das bibliotecas compartilhadas são a redução do espaço de memória e de disco utilizado por um sistema e a maior facilidade de manutenção.

Quando bibliotecas compartilhadas estão sendo utilizadas por todos os programas em execução, há somente uma instância de cada rotina da biblioteca do sistema em disco e, no máximo, uma instância na memória física. Quando a biblioteca em questão é utilizada por muitas aplicações e programas, então as economias de disco e memória podem ser bastante substanciais. Além disso, o tempo de início para a execução de novos programas pode ser reduzido, pois muitas das funções comuns necessárias ao programa provavelmente já estarão carregadas na memória física.

A facilidade de manutenção também é uma vantagem importante da vinculação dinâmica em relação à estática. Se todos os programas em execução utilizarem uma biblioteca compartilhada para acessar suas rotinas da biblioteca do sistema, então a melhoria dessas rotinas, seja para adicionar nova funcionalidade, seja para corrigir bugs, poderá ser feita simplesmente pela substituição da biblioteca compartilhada. Não há necessidade de recom-

pilar ou revincular quaisquer aplicações; qualquer programa carregado após a realização da melhoria acessará automaticamente as novas versões das bibliotecas.

Existem também outras vantagens. Um programa que utilize bibliotecas compartilhadas pode ser, com frequência, adaptado a finalidades específicas simplesmente substituindo uma ou mais de suas bibliotecas ou mesmo adicionando uma nova biblioteca em tempo de execução (se o sistema permitir isso; a maioria dos sistemas UNIX, incluindo o Linux, permite). Por exemplo, uma biblioteca de depuração pode ser substituída por uma biblioteca normal para rastrear um problema em uma aplicação. Bibliotecas compartilhadas também permitem que programas binários sejam vinculados a códigos de bibliotecas proprietárias e comerciais sem realmente incluir tais códigos no arquivo executável final do programa. Isso é importante porque, na maioria dos sistemas UNIX, muitas das bibliotecas compartilhadas padrão são proprietárias, e questões de licenciamento podem impedir a inclusão daquele código em arquivos executáveis a serem distribuídos a terceiros.

Em alguns locais, entretanto, a vinculação estática é apropriada. Um exemplo é em ambientes de recuperação para administradores de sistemas. Se um administrador de sistema cometer um erro ao instalar novas bibliotecas, ou no caso de o hardware desenvolver problemas, é bem possível que as bibliotecas compartilhadas existentes fiquem corrompidas. Como resultado, um conjunto básico de recursos de recuperação é frequentemente vinculado estaticamente, de modo que haja uma oportunidade de corrigir a falha sem ter que se valer de bibliotecas compartilhadas que estejam funcionando corretamente.

Existem também vantagens de desempenho que, algumas vezes, tornam a vinculação estática preferível em casos especiais. Para começar, a vinculação dinâmica aumenta o tempo de início de um programa, já que a vinculação deve ser feita, agora, em tempo de execução e não em tempo de compilação. A vinculação dinâmica também pode, algumas vezes, aumentar o tamanho máximo do conjunto de trabalho de um programa (o número total de páginas físicas de memória requeridas para executar o programa). Em uma biblioteca compartilhada, o usuário não tem controle sobre o local em que as várias funções residem no arquivo binário da biblioteca. Como a maioria das funções não preenche, precisamente, uma página ou páginas inteiras da biblioteca, a carga de uma função resultará normalmente na carga de partes das funções circunvizinhas também. Com a vinculação estática, absolutamente nenhuma função que não for referenciada (direta ou indiretamente) pela aplicação precisará ser carregada na memória.

Outras questões em torno da vinculação estática incluem a facilidade de distribuição: é mais fácil distribuir um arquivo executável com vinculação estática do que com vinculação dinâmica, se o distribuidor não estiver certo de que o receptor terá as bibliotecas corretas instaladas com antecedência. Também podem existir restrições comerciais contra a redistribuição de alguns binários como bibliotecas compartilhadas. Por exemplo, a licença para o ambiente gráfico "Motif" do UNIX permite que binários que utilizem o Motif sejam distribuídos livremente enquanto forem vinculados estaticamente, mas as bibliotecas compartilhadas não podem ser utilizadas sem licença.

- 18.5** Compare o uso de sockets de rede com o uso de memória compartilhada como mecanismo para a comunicação de dados entre processos em um único computador. Quais são as vantagens de cada método? Quando cada um é preferível?

Resposta:

A utilização de sockets de rede em vez de memória compartilhada para comunicação local tem muitas vantagens. A principal vantagem é que a interface de programação de sockets representa um rico conjunto de recursos de sincronização. Um processo pode determinar facilmente quando chegaram novos dados em uma conexão de socket, quantos dados estão presentes e quem os enviou. Os processos podem bloquear, até que cheguem novos dados em um socket, ou podem solicitar que seja liberado um sinal quando os dados chegarem. Um socket também gerencia conexões separadas. Um processo com um socket aberto para recepção pode aceitar múltiplas conexões para o socket e será informado quando novos processos tentarem se conectar, ou processos antigos liberarem suas conexões.

A memória compartilhada não oferece nenhum desses recursos. Não há modo de um processo determinar se outro processo liberou dados para a memória compartilhada ou alterou dados nela residentes, a não ser indo olhar o conteúdo da memória. É impossível para um processo bloquear e solicitar uma ativação quando a memória compartilhada for liberada, e não existe mecanismo-padrão para outros processos estabelecerem um link de memória compartilhada para um processo existente.

Entretanto, a memória compartilhada tem a vantagem de ser muito mais rápida do que as comunicações por socket, em muitos casos. Quando são enviados dados sobre um socket, eles normalmente são copiados de memória a memória múltiplas vezes. As atualizações na memória compartilhada não requerem cópias dos dados: se um processo atualizar uma estrutura de dados na memória compartilhada, essa atualização será imediatamente visível por todos os demais processos que estiverem compartilhando a memória. O envio e a recepção de dados sobre um socket requerem que seja feita uma chamada de serviço de sistema do kernel para iniciar a transferência, mas a comunicação por meio de memória compartilhada pode ser executada inteiramente em modalidade de usuário, sem exigir transferência de controle.

A comunicação por sockets é normalmente preferida quando o gerenciamento da conexão é importante, ou quando existe um requisito para sincronizar o emissor e o receptor. Por exemplo, processos servidores estabelecerão usualmente um socket de escuta com o qual os clientes possam se conectar quando quiserem utilizar o serviço. Uma vez que o socket seja estabelecido, as solicitações individuais também serão enviadas com o uso do socket, de modo que o servidor possa determinar facilmente quando chegar uma nova solicitação e de quem ela provém.

Em alguns casos, entretanto, a memória compartilhada é preferida. Ela é, com frequência, uma solução melhor quando grandes volumes de dados devem ser transferidos, ou quando dois processos precisam de acesso aleatório a um grande data set comum. Nesse caso, porém, os processos em comunicação podem precisar ainda de um mecanismo adicional à memória compartilhada para conseguir a sincronização entre eles. O X Window Sys-

tem, um ambiente de display gráfico do UNIX, é um bom exemplo disso: a maioria das solicitações de gráficos é enviada sobre sockets, mas a memória compartilhada é oferecida como um transporte adicional em casos especiais em que grandes mapas de bits devem ser exibidos em tela. Nesse caso, será enviada uma solicitação para exibir o mapa de bits sobre o socket, mas o volume de dados do próprio mapa de bits será transmitido via memória compartilhada.

- 18.6 Antigamente, os sistemas UNIX utilizavam otimizações do formato de discos baseadas na posição de rotação dos dados no disco, mas implementações modernas, incluindo o Linux, simplesmente realizam a otimização para acesso sequencial aos dados. Por que fazem isso? De que características de hardware o acesso sequencial tira vantagem? Por que a otimização rotacional não é mais tão útil?

Resposta:

As características de desempenho do hardware de disco têm mudado substancialmente nos últimos anos. Em particular, muitas melhorias têm sido introduzidas para otimizar a largura de banda máxima que pode ser obtida em um disco. Em um sistema moderno, pode existir um longo canal entre o sistema operacional e o cabeçote de leitura-gravação do disco. Uma solicitação de I/O de disco tem que passar pelo controlador de disco local do computador, sobre o bus para o próprio drive do disco e, então, internamente para o disco, em que existe provavelmente um controlador complexo que pode armazenar em cache os acessos aos dados e potencialmente otimizar a ordem das solicitações de I/O.

Por causa dessa complexidade, o tempo necessário para que uma solicitação de I/O seja reconhecida e a próxima solicitação seja gerada e recebida pelo disco pode superar em muito o montante de tempo entre a passagem de um setor do disco sob o cabeçote de leitura-gravação e a chegada do cabeçalho do próximo setor. Para serem capazes de ler eficientemente múltiplos setores de uma vez, os discos empregarão um cache de leitura-adiante. Enquanto um setor está sendo passado de volta ao computador hospedeiro, o disco estará ocupado lendo os próximos setores antes que uma solicitação os leia. Se as solicitações de leitura começarem a chegar em uma ordem que quebre esse canal de leitura-adiante, o desempenho cairá. Como resultado, o desempenho será substancialmente beneficiado, se o sistema operacional tentar manter as solicitações de I/O em ordem estritamente sequencial.

Uma segunda característica dos discos modernos é que sua geometria pode ser muito complexa. O número de setores por cilindro pode variar de acordo com a posição do cilindro: mais dados podem ser acomodados nas trilhas mais longas próximas à margem do disco do que no centro do disco. Para que um sistema operacional otimize a posição rotacional dos dados nesses discos, ele teria que deter o conhecimento completo da sua geometria, bem como das características de timing do disco e do seu controlador. Em geral, apenas a lógica interna do disco pode determinar o scheduling ótimo de I/Os, e a geometria do disco provavelmente descartará qualquer tentativa de execução de otimizações rotacionais pelo sistema operacional.



Windows 7

Exercícios Práticos

- 19.1 Que tipo de sistema operacional é o Windows 7? Descreva duas de suas principais características.

Resposta:

É um sistema operacional multitarefa preemptivo de 32/64 bits que suporta múltiplos usuários. (1) A capacidade de reparar automaticamente problemas da aplicação e do sistema operacional. (2) Melhor conexão em rede e experiência com dispositivos (incluindo fotografia digital e vídeo).

- 19.2 Liste os objetivos de projeto do Windows 7. Descreva dois deles com detalhes.

Resposta:

Os objetivos de projeto incluem: segurança, confiabilidade, compatibilidade com aplicações Window e POSIX, alto desempenho, extensibilidade, portabilidade e suporte internacional. (1) A confiabilidade foi percebida como um requisito restritivo e incluiu a verificação extensiva dos drivers, recursos para a captura de erros de programação em código de nível de usuário, e um processo rigoroso de certificação de drivers, aplicações e dispositivos de terceiros. (2) O alcance do alto desempenho exigiu o exame de áreas de problemas do passado, tais como o desempenho de I/O, os gargalos da CPU de servidores e a escalabilidade de ambientes multithreads e multiprocessadores.

- 19.3 Descreva o processo de inicialização de um sistema Windows 7.

Resposta:

(1) Enquanto o hardware é ligado, a BIOS começa a executar a partir da ROM e carrega e executa o carregador de bootstrap do disco. (2) O programa NTLDR é carregado a partir do diretório raiz do dispositivo do sistema identificado e determina qual dispositivo de inicialização contém o sistema operacional. (3) O NTLDR carrega a biblioteca HAL, o kernel e o hive do sistema. O hive do sistema indica os drivers de inicialização requeridos e os carrega. (4) A execução do kernel começa pela inicialização do sistema e pela criação de dois processos: o processo do sistema que contém todos os threads de trabalho internos e o primeiro processo de inicialização em modalidade de usuário, o SMSS. (5) O SMSS, então, inicializa o sistema estabelecendo os arquivos de paginação e carregando os drivers de dispositivos. (6) O SMSS cria dois processos: o WINLOGON, que traz o restante do sistema, e o CSRSS (o processo do subsistema Win32).

- 19.4 Descreva as três principais camadas de arquitetura do kernel do Windows 7.

Resposta:

- A camada de abstração do hardware (HAL — *Hardware Abstraction Layer*) cria a portabilidade do sistema operacional ocultando diferenças de hardware das camadas superiores do sistema operacional. Detalhes administrativos dos recursos de baixo nível são fornecidos pelas interfaces da HAL. A HAL apresenta uma interface de máquina virtual que é utilizada pelo despachante do kernel, pelo executivo e pelos drivers de dispositivos.
- A camada do kernel fornece uma base para as funções executivas e os subsistemas de modalidade de usuário. O kernel permanece em memória e nunca sofre preempção. Suas responsabilidades são o scheduling de threads, a manipulação de interrupções e exceções, a sincronização de processadores de baixo nível, e a recuperação de falhas de energia.
- A camada executiva fornece um conjunto de serviços utilizados por todos os subsistemas: gerenciador de objetos, gerenciador da memória virtual, gerenciador de processos, recursos de chamadas de procedimentos locais, gerenciador de I/O, monitor de segurança, gerenciador de plug-and-play, registro e inicialização.

- 19.5 Qual é a função do gerenciador de objetos?

Resposta:

Os objetos apresentam um conjunto genérico de interfaces em modalidade de kernel para programas em modalidade de usuário. Objetos são manipulados pelo gerenciador de objetos da camada executiva. A tarefa do gerenciador de objetos é supervisionar a alocação e o uso de todos os objetos gerenciados.

- 19.6 Que tipos de serviços o gerenciador de processos oferece?

Resposta:

O gerenciador de processos fornece serviços para a criação, exclusão e utilização de processos, threads e jobs. O gerenciador de processos também implementa o enfileiramento e a liberação de chamadas de procedimentos assíncronas para os threads.

- 19.7 O que é uma chamada de procedimento local?

Resposta:

A chamada de procedimento local (LPC) é um sistema de passagem de mensagens. O sistema operacional utiliza a LPC para passar solicitações e resultados entre processos

cliente e servidor dentro de uma única máquina, em particular entre subsistemas do Windows 7.

- 19.8** Quais são as responsabilidades do gerenciador de I/O?

Resposta:

O gerenciador de I/O é responsável pelos sistemas de arquivos, drivers de dispositivos e drivers de rede. O gerenciador de I/O controla quais drivers de dispositivos, drivers de filtro e sistemas de arquivos estão carregados, e gerencia os buffers para as solicitações de I/O. Além disso, toma parte do fornecimento do I/O de arquivos mapeados na memória e controla o gerenciador de caches para todo o sistema de I/O.

- 19.9** Que tipos de conexão de rede o Windows 7 suporta? Como o Windows 7 implementa protocolos de transporte? Descreva dois protocolos de conexão de rede.

Resposta:

O suporte é fornecido tanto para a conexão em rede cliente-servidor quanto para a peer-to-peer. Os protocolos de transporte são implementados como drivers. (1) O pacote TCP/IP inclui suporte do SNMP, DHCP, WINS e Net-BIOS. (2) O protocolo de tunneling ponto a ponto é fornecido para a comunicação entre módulos de acesso remoto, em execução em servidores Windows 7, e outros sistemas clientes conectados pela web. Utilizando esse esquema, redes virtuais privadas (VPNs) multiprotocolos são suportadas na web.

- 19.10** Como é organizado o espaço de nomes no NTFS?

Resposta:

O espaço de nomes no NTFS é organizado como uma hierarquia de diretórios em que cada diretório utiliza uma estrutura de dados em árvore B+ para armazenar um índice dos nomes dos arquivos naquele diretório. O índice raiz de um diretório contém o nível de topo da árvore B+. Cada entrada no diretório contém o nome e a referência de arquivo do arquivo, bem como o marcador de tempo de atualização e o tamanho do arquivo.

- 19.11** Como o NTFS manipula estruturas de dados? Como o NTFS se recupera de uma queda do sistema? O que é garantido após a ocorrência de uma recuperação?

Resposta:

No NTFS, todas as atualizações das estruturas de dados do sistema de arquivos são executadas dentro de transa-

ções. Antes que uma estrutura de dados seja alterada, a transação grava um registro de log contendo informações redo (refaça) e undo (desfaça). Um registro de confirmação é gravado no log após a ocorrência da transação. Após uma queda, o sistema de arquivos pode ser restaurado a um estado consistente por meio do processamento dos registros do log, primeiro refazendo as operações para as transações confirmadas e, depois, desfazendo as operações para as transações que não foram confirmadas com sucesso. Esse esquema não garante que o conteúdo dos arquivos de usuário esteja correto após uma recuperação, mas sim que as estruturas de dados do sistema de arquivos (metadados dos arquivos) não estão danificadas e refletem algum estado consistente que existia antes da queda.

- 19.12** Como o Windows 7 aloca a memória do usuário?

Resposta:

A memória do usuário pode ser alocada de acordo com diversos esquemas: memória virtual, arquivos mapeados na memória, heaps e armazenamento local de threads.

- 19.13** Descreva algumas das maneiras pelas quais uma aplicação pode utilizar memória por meio da API Win32.

Resposta:

- a. A memória virtual fornece diversas funções que permitem que uma aplicação reserve e libere memória, especificando o endereço virtual ao qual a memória está alocada.
- b. Um arquivo pode ser mapeado em memória no espaço de endereçamento, fornecendo um meio para que dois processos compartilhem memória.
- c. Ao ser inicializado, um processo Win32 é criado com um heap default. Podem ser criados heaps privados que fornecem regiões de espaço de endereçamento reservadas para as aplicações. São fornecidas funções de gerenciamento de threads para alocar e controlar o acesso dos threads aos heaps privados.
- d. Um mecanismo de armazenamento local de threads fornece um modo para que dados globais e estáticos funcionem adequadamente em um ambiente multithread. O armazenamento local de threads aloca memória global por thread.

Sistemas Operacionais Influentes



Não existem Exercícios Práticos.