Arquitetura de Sistemas Operacionais

3ª Edição

Versão: 3.2 (Jan/2005)

Francis Berenger Machado
Luiz Paulo Maia

Soluções dos Exercícios

LTC

Capítulo 1 – Visão Geral

1. Sem o sistema operacional, um usuário para interagir com o computador deveria conhecer profundamente diversos detalhes sobre hardware do equipamento, o que tornaria seu trabalho lento e com grandes possibilidades de erros. As duas principais funções são "facilidade de acesso aos recursos do sistema" e "compartilhamento de recursos de forma organizada e protegida".

2. O computador pode ser visualizado como uma máquina de camadas, onde inicialmente existem duas camadas: hardware (nível 0) e sistema operacional (nível 1). Desta forma, o usuário pode enxergar a máquina como sendo apenas o sistema operacional, ou seja, como se o hardware não existisse. Esta visão modular e abstrata é chamada máquina virtual. A vantagem desse conceito é tornar a interação entre usuário e computador mais simples, confiável e eficiente.

3. O computador pode ser visualizado como uma máquina de níveis ou máquina de camadas, possuindo tantos níveis quanto forem necessários para adequar o usuário às suas diversas aplicações. Quando o usuário está trabalhando em um desses níveis, não necessita saber da existência das outras camadas. Com isso a interação entre usuário e computador apresenta-se mais simples, confiável e eficiente.

4. Sistemas monoprogramáveis ou monotarefa, sistemas multiprogramáveis ou multitarefa e sistemas com múltiplos processadores.

5. Porque em sistemas monoprogramáveis somente é possível a execução de um programa por vez. Como um programa não utiliza todos os recursos do sistema totalmente ao longo da sua execução, existe ociosidade e, consequentemente, subutilização de alguns recursos.

6. Os sistemas monoprogramáveis se caracterizam por permitir que o processador, a memória e os periféricos permaneçam exclusivamente dedicados à execução de um único programa. Nos sistemas multiprogramáveis ou multitarefa, os recursos computacionais são compartilhados entre os diversos usuários e aplicações. Enquanto em sistemas monoprogramáveis existe apenas um programa utilizando os recursos disponíveis, nos multiprogramáveis várias aplicações compartilham esses mesmos recursos.

7. As vantagens do uso de sistemas multiprogramáveis são a redução do tempo de resposta das aplicações processadas no ambiente e de custos, a partir do compartilhamento dos diversos recursos do sistema entre as diferentes aplicações.

8. Sim, somente um usuário interage com o sistema podento possuir diversas aplicações executando concorrentemente. O sistema Windows NT é um exemplo.

9. Sistemas batch, sistemas de tempo compartilhado e sistemas de tempo real.

10. O processamento batch tem a característica de não exigir a interação do usuário com a aplicação. Todas as entradas e saídas de dados da aplicação são implemetadas por algum tipo de memória secundária, geralmente arquivos em disco. Alguns exemplos de aplicações originalmente processadas em batch são programas envolvendo cálculos numéricos, compilações, ordenações, backups e todos aqueles onde não é necessária a interação com o usuário.

11. Os sistemas de tempo compartilhado (time-sharing) permitem que diversos programas sejam executados a partir da divisão do tempo do processador em pequenos intervalos, denomidados fatia de tempo (time-slice). A vantagem na sua utilização é possibilitar para cada usuário um ambiente de trabalho próprio, dando a impressão de que todo o sistema está dedicado, exclusivamente, a ele.

12. O fator tempo de resposta. Nos sistemas de tempo real, os tempos de resposta devem estar dentro de limites rígidos. Aplicações de controle de processos, como no monitoramento de refinarias de petróleo, controle de tráfego aéreo, de usinas termoelétricas e nucleares são executadas em sistemas de tempo real.

13. Os sistemas com múltiplos processadores caracterizam-se por possuir duas ou mais UCPs interligadas e trabalhando em conjunto. A vantagem deste tipo de sistema é permitir que vários programas sejam executados ao mesmo tempo ou que um mesmo programa seja subdividido em partes para serem executadas simultaneamente em mais de um processador.

14. Nos sistemas fortemente acoplados existem vários processadores compartilhando uma única memória física e dispositivos de entrada/saída, sendo gerenciados por apenas um sistema operacional. Os sistemas fracamente acoplados caracterizam-se por possuir dois ou mais sistemas computacionais conectados através de linhas de comunicação. Cada sistema funciona de forma independente, possuindo seu próprio sistema operacional e gerenciando seus próprios recursos, como UCP, memória e dispositivos de entrada/saída.

15. Nos sistemas SMP, o tempo de acesso à memória principal pelos diversos processadores é uniforme. Nos sistemas NUMA, existem diversos conjuntos de processadores e memória principal interconectados, onde o tempo de acesso à memória principal varia em função da sua localização física.

16. Os sistemas fracamente acoplados caracterizam-se por possuir dois ou mais sistemas computacionais conectados através de linhas de comunicação. Cada sistema funciona de forma independente, possuindo seu próprio sistema operacional e gerenciando seus próprios recursos, como UCP, memória e dispositivos de entrada/saída. Os sistemas operacionais de rede permitem que um host compartilhe seus recursos, como uma impressora ou diretório, com os demais hosts da rede enquanto que nos sistemas distribuídos, o sistema operacional esconde os detalhes dos hosts individuais e passa a tratá-los como um conjunto único, como se fosse um sistema fortemente acoplado.

Capítulo 2 – Conceitos de Hardware e Software

1. Processador ou unidade central de processamento, memória principal e dispositivos de entrada/saída.

2. Um processador é composto por unidade de controle, unidade lógica e aritmética, e registradores. A unidade de controle (UC) é responsável por gerenciar as atividades de todos os componentes do computador, como a gravação de dados em discos ou a busca de instruções na memória. A unidade lógica e aritmética (ULA), como o nome indica, é responsável pela realização de operações lógicas (testes e comparações) e aritméticas (somas e subtrações).

3. A memória é composta por unidades de acesso chamadas células, sendo cada célula composta por um determinado número de bits. Atualmente, a grande maioria dos computadores utiliza o byte (8 bits) como tamanho de célula.

4. No ciclo de leitura, a UCP armazena no MAR, o endereço da célula a ser lida e gera um sinal de controle para a memória principal, indicando que uma operação de leitura deve ser realizada. O conteúdo da(s) célula(s), identificada(s) pelo endereço contido no MAR, é transferido para o MBR

No cliclo de gravação, a UCP armazena no MAR, o endereço da célula que será gravada e armazena no MBR, a informação que deverá ser gravada. A UCP gera um sinal de controle para a memória principal, indicando que uma operação de gravação deve ser realizada e a informação contida no MBR é transferida para a célula de memória endereçada pelo MAR

5.
MAR=16 bits número max células = 2^{16}
MAR=32 bits número max células = 2^{32}
MAR=64 bits número max células = 2^{64}

6. Memórias voláteis precisam estar sempre energizadas para manter suas informações, o que não acontece com as não-voláteis.

7. A memória cache é uma memória volátil de alta velocidade, porém com pequena capacidade de armazenamento. O tempo de acesso a um dado nela contido é muito menor que se o mesmo estivesse na memória principal. O propósito do uso da memória cache é minimizar a disparidade existente entre a velocidade com que o processador executa instruções e a velocidade com que dados são acessados na memória principal.

8. A memória principal é um dispositivo de armazenamento, em geral volátil, onde são armazenados instruções e dados utilizados pelo processador durante a execução de programas. A memória secundária é um dispositivo não-volátil com maior capacidade de armazenamento, porém com menor velocidade de acesso aos seus dados armazenados.

9. Os dispositivos de entrada e saída podem ser divididos em duas categorias: os que são utilizados como memória secundária e os que servem para a interface usuário-máquina. Os dispositivos utilizados como memória secundária (discos e fitas magnéticas) caracterizam-se por ter capacidade de armazenamento bastante superior ao da memória principal. Seu custo é relativamente baixo, porém o tempo de acesso à memória secundária é bem superior ao da memória principal. Outros dispositivos têm como finalidade a comunicação usuário-máquina, como teclados, monitores de vídeo, impressoras e plotters.

10. Os barramentos processador-memória são de curta extensão e alta velocidade para que seja otimizada a transferência de informação entre processadores e memórias. Os barramentos de E/S possuem maior extensão, são mais lentos e permitem a conexão de diferentes dispositivos. O barramento de backplane tem a função de integrar os dois barramentos anteriores.

11. Permitindo ao processador executar múltiplas instruções paralelamente em estágios diferentes.

12. Ver Tabela 2.3 do livro.

13. A técnica conhecida como benchmark permite a análise de desempenho comparativa entre sistemas computacionais. Neste método, um conjunto de programas é executado em cada sistema avaliado e o tempo de execução comparado. A escolha dos programas deve ser criteriosa para refletir os diferentes tipos de aplicação.

14. Isso ocorre em função de um programa poder chamar sub-rotinas externas, e, neste caso, o tradutor não tem como associar o programa principal às sub-rotinas chamadas. Esta função é realizada pelo linker.

15. Como não existe a geração de um código executável, as instruções de um programa devem ser traduzidas toda vez que este for executado.

16. Suas funções básicas são resolver todas as referências simbólicas existentes entre os módulos de um programa e reservar memória para sua execução.

17. Carregar na memória principal um programa para ser executado.

18. O depurador oferece ao usuário recursos como acompanhar a execução de um programa instrução por instrução; possibilitar a alteração e visualização do conteúdo de variáveis; implementar pontos de parada dentro do programa (breakpoint), de forma que, durante a execução, o programa pare nesses pontos e especificar que, toda vez que o conteúdo de uma variável for modificado, o programa envie uma mensagem (watchpoint).

19. Pesquisa livre.

20. Inicialmente, todo o código do sistema operacional reside memória secundária como discos e fitas. Toda vez que um computador é ligado, o sistema operacional tem que ser carregado da memória secundária para a memória principal. Esse procedimento é realizado por um programa localizado em um bloco específico do disco (boot block).

Capítulo 3 - Concorrência

1. Concorrência é o princípio básico para projeto e implementação dos sistemas operacionais multiprogramáveis onde é possível o processador executar instruções em paralelo com operações de E/S. Isso possibilita a utilização concorrente da UCP por diversos programas sendo implementada de maneira que, quando um programa perde o uso do processador e depois retorna para continuar o processamento, seu estado deve ser idêntico ao do momento em que foi interrompido. O programa deverá continuar sua execução exatamente na instrução seguinte àquela em que havia parado, aparentando ao usuário que nada aconteceu.

2. Porque é em função desse mecanismo que o sistema operacional sincroniza a execução de todas as suas rotinas e dos programas dos usuários, além de controlar dispositivos.

3. Uma interrupção é sempre gerada por algum evento externo ao programa e, neste caso, independe da instrução que está sendo executada. Ao final da execução de cada instrução, a unidade de controle verifica a ocorrência de algum tipo de interrupção. Neste caso, o programa em execução é interrompido e o controle desviado para uma rotina responsável por tratar o evento ocorrido, denominada rotina de tratamento de interrupção. Para que o programa possa posteriormente voltar a ser executado, é necessário que, no momento da interrupção, um conjunto de informações sobre a sua execução seja preservado. Essas informações consistem no conteúdo de registradores, que deverão ser restaurados para a continuação do programa.

4. Evento síncronos são resultados direto da execução do programa corrente. Tais eventos são previsíveis e, por definição, só podem ocorrer um único de cada vez. Eventos assíncronos não são relacionados à instrução do programa corrente. Esses eventos, por serem imprevisíveis, podem ocorrer múltiplas vezes, como no caso de diversos dispositivos de E/S informarem ao processador que estão prontos para receber ou transmitir dados. Uma interrupção é um evento assíncrono enquanto uma exceção é um evento síncrono.

5. Uma instrução que gere a situação de overflow ou uma divisão por zero.

6. Na E/S controlada por interrupção, as operações de E/S podem ser realizadas de uma forma mais eficiente. Em vez de o sistema periodicamente verificar o estado de uma operação pendente como na técnica de polling, o próprio controlador interrompe o processador para avisar do término da operação. Com esse mecanismo, o processador, após a execução de um comando de leitura ou gravação, permanece livre para o processamento de outras tarefas.

7. A técnica de DMA permite que um bloco de dados seja transferido entre a memória principal e dispositivos de E/S, sem a intervenção do processador, exceto no início e no final da transferência. Quando o sistema deseja ler ou gravar um bloco de dados, o processador informa ao controlador sua localização, o dispositivo de E/S, a posição inicial da memória de onde os dados serão lidos ou gravados e o tamanho do bloco. Com estas informações, o controlador realiza a transferência entre o periférico e a memória principal, e o processador é somente interrompido no final da operação.

8. Como o buffering permite minimizar o problema da disparidade da velocidade de processamento existente entre o processador e os dispositivos de E/S, esta técnica permite manter, na maior parte do tempo, processador e dispositivos de E/S ocupados.

9. No momento em que um comando de impressão é executado, as informações que serão impressas são gravadas antes em um arquivo em disco, conhecido como arquivo de spool, liberando imediatamente o programa para outras atividades. Posteriormente, o sistema operacional encarrega-se em direcionar o conteúdo do arquivo de spool para a impressora.

10. Sem reentrância, cada usuário teria sua cópia do código na memória totalizando 10 x (200 Kb + 300 Kb + 200 Kb + 500 Kb) = 12.000 Kb. Caso a reentrância seja implementada, apenas uma cópia do código seria necessária na memória principal (200 Kb + 300 Kb + 200 Kb + 500 Kb) totalizando 1.200 Kb. Um total de 10.800 Kb seriam liberados da memória principal.

11. Se considerarmos que diversos usuários estão compartilhando os mesmos recursos como memória, processador e dispositivos de E/S, deve existir uma preocupação em garantir a confiabilidade e a integridade dos programas e dados dos usuários, além do próprio sistema operacional.

Capítulo 4 – Estrutura do Sistema Operacional

1. É o conjunto de rotinas que oferece serviços aos usuários, suas aplicações, além do próprio sistema operacional. As principais funções do núcleo encontradas na maioria dos sistemas comerciais são: tratamento de interrupções e exceções; criação e eliminação de processos e threads; sincronização e comunicação entre processos e threads; escalonamento e controle dos processos e threads; gerência de memória; gerência do sistema de arquivos; gerência de dispositivos de E/S; suporte à redes locais e distribuídas; contabilização do uso do sistema; auditoria e segurança do sistema.

2. As system calls podem ser entendidas como uma porta de entrada para o acesso ao núcleo do sistema operacional e a seus serviços. Sempre que um usuário ou aplicação desejar algum serviço do sistema, é realizada uma chamada a uma de suas rotinas através de uma system call. Através dos parâmetros fornecidos na system call, a solicitação é processada e uma resposta é retornada a aplicação juntamente com um estado de conclusão indicando se houve algum erro. O mecanismo de ativação e comunicação entre o programa e o sistema operacional é semelhante ao mecanismo implementado quando um programa chama uma subrotina.

3. Instruções privilegiadas são instruções que só devem ser executadas pelo sistema operacional ou sob sua supervisão, impedindo, assim, a ocorrência de problemas de segurança e integridade do sistema. As instruções não-privilegiadas não oferecem risco ao sistema. Quando o processador trabalha no modo usuário, uma aplicação só pode executar instruções não-privilegiadas, tendo acesso a um número reduzido de instruções, enquanto no modo kernel ou supervisor a aplicação pode ter acesso ao conjunto total de instruções do processador.

4. Desabilitar todas as interrupções, alterar a data e hora do sistema, alterar informações residentes no núcleo do sistema e acessar diretamente posições no disco.

5. Sempre que um programa necessita executar uma instrução privilegiada, a solicitação deve ser realizada através de uma chamada a uma system call, que altera o modo de acesso do processador do modo usuário para o modo kernel. Ao término da execução da rotina do sistema, o modo de acesso retorna para o modo usuário.

6. Através do modo de acesso de uma aplicação determinado por um conjunto de bits localizado no registrador de status do processador ou PSW. Através desse registrador, o hardware verifica se a instrução pode ou não ser executada pela aplicação, possibilitando proteger o kernel do sistema operacional de um acesso indevido.

7. A arquitetura monolítica pode ser comparada com uma aplicação formada por vários módulos que são compilados separadamente e depois linkados, formando um grande e único programa executável, onde os módulos podem interagir livremente. Na arquitetura de camadas, o sistema é dividido em níveis sobrepostos. Cada camada oferece um conjunto de funções que podem ser utilizadas apenas pelas camadas superiores. A vantagem da estruturação em camadas é isolar as funções do sistema operacional, facilitando sua manutenção e depuração, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas. Uma desvantagem para o modelo de camadas é o desempenho. Cada nova camada implica em uma mudança no modo de acesso.

8. Além de permitir a convivência de sistemas operacionais diferentes no mesmo computador, a vantagem desse modelo é criar um isolamento total entre cada VM, oferecendo grande segurança para cada máquina virtual.

9. Sempre que uma aplicação deseja algum serviço, é realizada uma solicitação ao processo responsável. Neste caso, a aplicação que solicita o serviço é chamada de cliente, enquanto o processo que responde à solicitação é chamado de servidor. Um cliente, que pode ser uma aplicação de um usuário ou um outro componente do sistema operacional, solicita um serviço enviando uma mensagem para o servidor. O servidor responde ao cliente através de uma outra mensagem. A utilização deste modelo permite que os servidores executem em modo usuário, ou seja, não tenham acesso direto a certos componentes do sistema. Apenas o núcleo do sistema, responsável pela comunicação entre clientes e servidores, executa no modo kernel. Como conseqüência, se ocorrer um erro em um servidor, este poderá parar, mas o sistema não ficará inteiramente comprometido, aumentando assim a sua disponibilidade. Outra vantagem é que a arquitetura microkernel permite isolar as funções do sistema operacional por diversos processos servidores pequenos e dedicados a serviços específicos, tornado o núcleo menor, mais fácil de depurar e, conseqüentemente, aumentando sua confiabilidade. Na arquitetura microkernel, o sistema operacional passa a ser de mais fácil manutenção, flexível e de maior portabilidade. Apesar de todas as vantagens deste modelo, sua implementação, na prática, é muito difícil. Primeiro existe o problema de desempenho, devido a necessidade de mudança de modo de acesso a cada

comunicação entre clientes e servidores. Outro problema é que certas funções do sistema operacional exigem acesso direto ao hardware, como operações de E/S.

10. Existe uma série de vantagens na utilização de programação por objetos no projeto e na implementação de sistemas operacionais. Os principais benefícios são: melhoria na organização das funções e recursos do sistema; redução no tempo de desenvolvimento; maior facilidade na manutenção e extensão do sistema; facilidade de implementação do modelo de computação distribuída.

Capítulo 5 – Processo

1. Um processo pode ser definido como o ambiente onde um programa é executado. Este ambiente, além das informações sobre a execução, possui também o quanto de recursos do sistema cada programa pode utilizar, como o espaço de endereçamento, tempo de processador e área em disco.

2. Através de processos, um programa pode alocar recursos, compartilhar dados, trocar informações e sincronizar sua execução. Nos sistemas multiprogramáveis os processos são executados concorrentemente, compartilhando o uso do processador, memória principal, dispositivos de E/S dentre outros recursos.

3. Sim, pois a execução de um programa pode necessitar de recursos do sistema que um processo pode possuir enquanto outro não.

4. Um processo é formado por três partes, conhecidas como contexto de hardware, contexto de software e espaço de endereçamento, que juntos mantêm todas as informações necessárias à execução de um programa.

5. O contexto de hardware armazena o conteúdo dos registradores gerais da UCP, além dos registradores de uso específico como program counter (PC), stack pointer (SP) e registrador de status. Quando um processo está em execução, o seu contexto de hardware está armazenado nos registradores do processador. No momento em que o processo perde a utilização da UCP, o sistema salva as informações no contexto de hardware do processo.

6. No contexto de software são especificadas características e limites dos recursos que podem ser alocados pelo processo, como o número máximo de arquivos abertos simultaneamente, prioridade de execução e tamanho do buffer para operações de E/S. O contexto de software é composto por três grupos de informações sobre o processo: identificação, quotas e privilégios. Ver item 5.2.2.

7. O espaço de endereçamento é a área de memória pertencente ao processo onde as instruções e dados do programa são armazenados para execução. Cada processo possui seu próprio espaço de endereçamento, que deve ser devidamente protegido do acesso dos demais processos.

8. O processo é implementado pelo sistema operacional através de uma estrutura de dados chamada bloco de controle do processo (Process Control Block — PCB). A partir do PCB, o sistema operacional mantém todas as informações sobre o contexto de hardware, contexto de software e espaço de endereçamento de cada processo.

9. Estado de Execução: processo que está sendo processado pela UCP no momento.
Estado de Pronto: processo que aguarda para ser executado.
Estado de Espera: processo que aguarda por algum evento ou recurso para prosseguir processamento.
Estado de Criação: processo cujo PCB já foi criado porém ainda não teve seu processamento iniciado.
Estado de Terminado: processo que não pode ter mais nenhum programa executado no seu contexto, porém o sistema operacional mantém suas informações de controle presentes na memória.

10. Livre.

11. Processos independentes não têm vínculo com os processos criadores. A criação de um processo independente exige a alocação de um PCB, possuindo contextos de hardware, contexto de software e espaço de endereçamento próprios.

Subprocessos são processos criados dentro de uma estrutura hierárquica. Caso um processo pai deixe de existir, os subprocessos subordinados são automaticamente eliminados. Semelhante aos processos independentes, subprocessos possuem seu próprio PCB. Além da dependência hierárquica entre processos e subprocessos, uma outra característica neste tipo de implementação é que subprocessos podem compartilhar quotas com o processo pai. Neste caso, quando um subprocesso é criado, o processo pai cede parte de suas quotas ao processo filho.

Processos multithreads suportam múltiplos threads, cada qual associado a uma parte do código da aplicação. Neste caso não é necessário haver diversos processos para a implementação da concorrência. Threads compartilham o processador da mesma maneira que um processo, ou seja, enquanto um thread espera por uma operação de E/S, outro thread pode ser executado.

12. Um processo foreground é aquele que permite a comunicação direta do usuário com o processo durante o seu processamento. Neste caso, tanto o canal de entrada quanto o de saída estão associados a um terminal com teclado, mouse e monitor, permitindo, assim, a interação com o usuário. Um processo background é aquele onde não existe a comunicação com o usuário durante o seu processamento. Neste caso, os canais de E/S não estão associados a nenhum dispositivo de E/S interativo, mas em geral a arquivos de E/S.

13. A arquitetura microkernel baseia-se na utilização de processos em modo usuário para executar diversas funções relativas ao sistema operacional, como gerência de memória e escalonamento.

14. Livre.

15. Quando ocorre uma divisão por zero, por exemplo, o sistema operacional é notificado do problema através de uma exceção. Por sua vez, o sistema deve notificar ao processo que gerou o problema através de um sinal.

16. Quando um processo é eliminado, o sistema ativa o sinal associado a este evento. O processo somente será excluído do sistema quando for selecionado para execução. Neste caso, é possível que o processo demore algum período de tempo até ser eliminado de fato.

Capítulo 6 – Thread

1. Através de processos independentes e subprocessos.

2. Um problema é que o uso de processos no desenvolvimento de aplicações concorrentes demanda consumo de diversos recursos do sistema. Sempre que um novo processo é criado, o sistema deve alocar recursos para cada processo, consumindo tempo de processador neste trabalho. No caso do término do processo, o sistema dispensa tempo para desalocar recursos previamente alocados.

Outro problema a ser considerado é quanto ao compartilhamento do espaço de endereçamento. Como cada processo possui seu próprio espaço de endereçamento, a comunicação entre processos torna-se difícil e lenta, pois utiliza mecanismos como pipes, sinais, semáforos, memória compartilhada ou troca de mensagem.

3. Um thread pode ser definido como uma subrotina de um programa que pode ser executada de forma assíncrona, ou seja, executada paralelamente ao programa chamador. A grande vantagem no uso de threads é a possibilidade de minimizar a alocação de recursos do sistema, além de diminuir o overhead na criação, troca e eliminação de processos.

4. Em ambientes monothread, o processo é ao mesmo tempo a unidade de alocação de recursos e a unidade de escalonamento. A independência entre os conceitos de processo e thread permite separar a unidade de alocação de recursos da unidade de escalonamento, que em ambientes monothread estão fortemente relacionadas. Em um ambiente multithread, a unidade de alocação de recursos é o processo, onde todos os seus threads compartilham o espaço de endereçamento, descritores de arquivos e dispositivos de E/S. Por outro lado, cada thread representa uma unidade de escalonamento independente e, neste caso, o sistema não seleciona um processo para a execução, mas sim um de seus threads.

5. Como threads de um mesmo processo compartilham o mesmo espaço de endereçamento, não existe qualquer proteção no acesso à memória, permitindo que um thread possa alterar facilmente dados de outros. Para que threads trabalhem de forma cooperativa, é fundamental que a aplicação implemente mecanismos de comunicação e sincronização entre threads, a fim de garantir o acesso seguro aos dados compartilhados na memória. Por outro lado, o compartilhamento do espaço de endereámento é extremamente simples e rápido.

6. Threads em modo usuário (TMU) são implementados pela aplicação e não pelo sistema operacional. Para isso, deve existir uma biblioteca de rotinas que possibilita à aplicação realizar tarefas como criação/eliminação de threads, troca de mensagens entre threads e uma política de escalonamento. Neste modo, o sistema operacional não sabe da existência de múltiplos threads, sendo responsabilidade exclusiva da aplicação gerenciar e sincronizar os diversos threads existentes.

Threads em modo kernel (TMK) são implementadas diretamente pelo núcleo do sistema operacional, através de chamadas a rotinas do sistema que oferecem todas as funções de gerenciamento e sincronização. O sistema operacional sabe da existência de cada thread e pode escaloná-los individualmente. No caso de múltiplos processadores, os threads de um mesmo processo podem ser executados simultaneamente.

7. A principal vantagem é melhorar o desempenho no seu uso evitando as mudanças de modos de acesso desnecessárias (usuário-kernel-usuário). Caso um thread utilize uma chamada ao sistema que o coloque no estado de espera, não é necessário que o kernel seja ativado, bastando que a própria biblioteca em modo usuário escalone outro thread. Isto é possível porque a biblioteca em modo usuário e o kernel se comunicam e trabalham de forma cooperativa. Cada camada implementa seu escalonamento de forma independente, porém trocando informações quando necessário.

8. Livre.

9. Para obter os benefícios do uso de threads, uma aplicação deve permitir que partes diferentes do seu código sejam executadas em paralelo de forma independente. O uso de uma arquitetura com múltiplos processasdores beneficia a concorrência entre os threads com a possibilidade do paralelismo de execução entre processadores.

10. O principal benefício do uso de threads em ambientes cliente-servidor é a melhoria no desempenho da aplicação servidora. Além disso, a comunicação entre os threads no servidor pode ser feita através de mecanismos mais simples e eficientes.

11. A arquitetura microkernel utiliza processos para implementar funções relativas ao kernel do sistema operacional, sendo que esses processos são utilizados como servidores quando algum cliente necessita de algum serviço do sistema. Arquiteturas que implementam threads, possibilitam um melhor desempenho dos processos servidores.

Capítulo 7 – Sincronização e Comunicação entre Processos

1. É uma aplicação estruturada de maneira que partes diferentes do código do programa possam executar concorrentemente. Este tipo de aplicação tem como base a execução cooperativa de múltiplos processos ou threads, que trabalham em uma mesma tarefa na busca de um resultado comum.

2. Caso não haja uma gerência no uso concorrente dos recursos compartilhados, inconsistências nos dados podem ocorrer.

3. É impedir que dois ou mais processos acessem um mesmo recurso simultaneamente. Para isso, enquanto um processo estiver acessando determinado recurso, todos os demais processos que queiram acessá-lo deverão esperar pelo término da utilização do recurso

4. Garantindo na aplicação que somente um único processo pode estar acessando a matriz por vez.

5. Starvation é a situação onde um processo nunca consegue executar sua região crítica e, conseqüentemente, acessar o recurso compartilhado. A solução para o problema depende de estabelecimentos de mecanismos de acesso pelo sistema operacional que garantam o acesso ao recurso por todos os processos que solicitarem uso.

6. Essa solução apesar de simples, apresenta algumas limitações. Primeiramente, a multiprogramação pode ficar seriamente comprometida, já que a concorrência entre processos tem como base o uso de interrupções. Um caso mais grave poderia ocorrer caso um processo desabilitasse as interrupções e não tornasse a habilitá-las. Nesse caso, o sistema, provavelmente, teria seu funcionamento seriamente comprometido.

Em sistemas com múltiplos processadores, esta solução torna-se ineficiente devido ao tempo de propagação quando um processador sinaliza aos demais que as interrupções devem ser habilitadas ou desabilitadas. Outra consideração é que o mecanismo de clock do sistema é implementado através de interrupções, devendo esta solução ser utilizada com bastante critério.

7. Na espera ocupada, toda vez que um processo não consegue entrar em sua região crítica, por já existir outro processo acessando o recurso, o processo permanece em looping, testando uma condição, até que lhe seja permitido o acesso. Dessa forma, o processo em looping consome tempo do processador desnecessariamente, podendo ocasionar problemas ao desempenho do sistema.

8. Sincronização condicional é uma situação onde o acesso ao recurso compartilhado exige a sincronização de processos vinculada a uma condição de acesso. Um recurso pode não se encontrar pronto para uso devido a uma condição específica. Nesse caso, o processo que deseja acessá-lo deverá permanecer bloqueado até que o recurso fique disponível. Um exemplo clássico desse tipo de sincronização é a comunicação entre dois processos através de operações de gravação e leitura em um buffer.

9. Um semáforo é uma variável inteira, não negativa, que só pode ser manipulada por duas instruções: DOWN e UP. Ver itens 7.7.1 e 7.7.2.

10. Livre.

11. Monitores são mecanismos de sincronização de alto nível que torna mais simples o desenvolvimento de aplicações concorrentes. Ver itens 7.8.1 e 7.8.2.

12. A vantagem deste mecanismo é aumentar a eficiência de aplicações concorrentes. Para implementar essa solução, além da necessidade de buffers para armazenar as mensagens, devem haver outros mecanismos de sincronização que permitam ao processo identificar se uma mensagem já foi enviada ou recebida.

13. Deadlock é a situação em que um processo aguarda por um recurso que nunca estará disponível ou um evento que não ocorrerá. Para que ocorra a situação de deadlock, quatro condições são necessárias simultaneamente:

- exclusão mútua: cada recurso só pode estar alocado a um único processo em um determinado instante;
- espera por recurso: um processo, além dos recursos já alocados, pode estar esperando por outros recursos;

- não-preempção: um recurso não pode ser liberado de um processo só porque outros processos desejam o mesmo recurso;
- espera circular: um processo pode ter de esperar por um recurso alocado a outro processo e vice-versa.

Para prevenir a ocorrência de deadlocks, é preciso garantir que uma das quatro condições apresentadas, necessárias para sua existência, nunca se satisfaça. A prevenção de deadlocks evitando-se a ocorrência de qualquer uma das quatro condições é bastante limitada e, por isso, na prática não é utilizada. Uma solução conhecida como Algoritmo do Banqueiro (implementada com a presença das quatro condições) também possui várias limitações. A maior delas é a necessidade de um número fixo de processos ativos e de recursos disponíveis no sistema. Essa limitação impede que a solução seja implementada na prática, pois é muito difícil prever o número de usuários no sistema e o número de recursos disponíveis.

14.
a) Cliente A = 200 e Cliente B = 1.200
b) Cliente A = 400 e Cliente B = 1.100
c)

Processo 1 (Cliente A)

```
/* saque em A */
Down (S1)
x := saldo_do_cliente_A;
x := x - 200;
saldo_do_cliente_A := x;
Up (S1)

/* deposito em B */
Down (S2)
x := saldo_do_cliente_B;
x := x + 100;
saldo_do_cliente_B := x;
Up (S2)
```

Processo 2 (Cliente B)

```
/*saque em A */
Down (S1)
y := saldo_do_cliente_A;
y := y - 100;
saldo_do_cliente_A := y;
Up (S1)

/* deposito em B */
Down (S2)
y := saldo_do_cliente_B;
y := y + 200;
saldo_do_cliente_B := y;
Up (S2)
```

15.
a) Acesso=0 Exclusao=1 Nleitores=1
b) Acesso=0 Exclusao=1 Nleitores=1, o bloqueio ocorre no semáforo Acesso.
c) Acesso=0 Exclusao=1 Nleitores=2
d) Não, pois a exclusão mútua a esta variável é implementada pelo semáforo Exclusao.
e) O processo Escritor inicia a escrita. Acesso=0 Exclusao=1 Nleitores=0
f) Os processo ficam bloqueados no semáforo Acesso. Acesso=0 Exclusao=0 Nleitores=1
g) Não, em geral os sistemas operacionais utilizam a escolha randômica dentre os processos em estado de espera.
h) Caso um processo Escritor esteja aguardando, bloqueado pelo semáforo Acesso, e sempre surgirem novos processos Leitor, o processo Escritor pode nunca ganhar acesso ao recurso.

Capítulo 8 – Gerência do Processador

1. Uma política de escalonamento é composta por critérios estabelecidos para determinar qual processo em estado de pronto será escolhido para fazer uso do processador.

2. O escalonador é uma rotina do sistema operacional que tem como principal função implementar os critérios da política de escalonamento. O dispatcher é responsável pela troca de contexto dos processos após o escalonador determinar qual processo deve fazer uso do processador.

3. Utilização do processador, throughput, tempo de Processador (tempo de UCP), tempo de espera, tempo de turnaround e tempo de resposta.

4. Tempo de processador ou tempo de UCP é o tempo que um processo leva no estado de execução durante seu processamento. Tempo de espera é o tempo total que um processo permanece na fila de pronto durante seu processamento, aguardando para ser executado. Tempo de turnaround é o tempo que um processo leva desde a sua criação até ao seu término, levando em consideração todo o tempo gasto na espera para alocação de memória, espera na fila de pronto (tempo de espera), processamento na UCP (tempo de processador) e na fila de espera, como nas operações de E/S. Tempo de resposta é o tempo decorrido entre uma requisição ao sistema ou à aplicção e o instante em que a resposta é exibida..

5. No escalonamento preemptivo, o sistema operacional pode interromper um processo em execução e passá-lo para o estado de pronto, com o objetivo de alocar outro processo na UCP. No escalonamento não-preemptivo, quando um processo está em execução, nenhum evento externo pode ocasionar a perda do uso do processador. O processo somente sai do estado de execução, caso termine seu processamento ou execute instruções do próprio código que ocasionem uma mudança para o estado de espera.

6. O FIFO é um escalonamento não-preemptivo onde o processo que chegar primeiro ao estado de pronto é o selecionado para execução. Este algoritmo é bastante simples, sendo necessária apenas uma fila, onde os processos que passam para o estado de pronto entram no seu final e são escalonados quando chegam ao seu início. Quando um processo vai para o estado de espera, o primeiro processo da fila de pronto é escalonado. Todos os processos quando saem do estado de espera entram no final da fila de pronto. O Circular é um escalonamento preemptivo, projetado especialmente para sistemas de tempo compartilhado. Esse algoritmo é bastante semelhante ao FIFO, porém, quando um processo passa para o estado de execução, existe um tempo limite para o uso contínuo do processador denominado fatia de tempo (time-slice) ou quantum.

7. No escalonamento SJF, o algoritmo de escalonamento seleciona o processo que tiver o menor tempo de processador ainda por executar. Dessa forma, o processo em estado de pronto que necessitar de menos tempo de UCP para terminar seu processamento é selecionado para execução. O escalonamento por prioridades é um escalonamento do tipo preemptivo realizado com base em um valor associado a cada processo denomidado prioridade de execução. O processo com maior prioridade no estado de pronto é sempre o escolhido para execução e processos com valores iguais são escalonados seguindo o critério de FIFO. Neste escalonamento, o conceito de fatia de tempo não existe, conseqüentemente, um processo em execução não pode sofrer preempção por tempo.

8. Preempção por tempo ocorre quando o sistema operacional interrompe o processo em execução em função da expiração da sua fatia de tempo, substituindo-o por outro processo. Preempção por prioridade, ocorre quando o sistema operacional interrompe o processo em execução em função de um processo entrar em estado de pronto com prioridade superior ao do processo em execução.

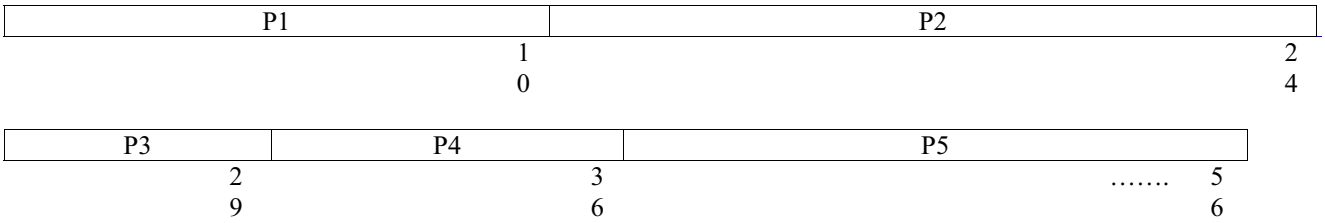
9. É um mecanismo onde o sistema operacional identifica o comportamento dos processos durante sua execução adaptando as políticas de escalonamento dinamicamente.

10. Escalonamento por prioridades onde é possível atribuir prioridades aos processos em função da sua importância. Além disso, o mecanismo de preempção por prioridades garante o escalonamento imediato de processos críticos quando esses passam para o estado de pronto.

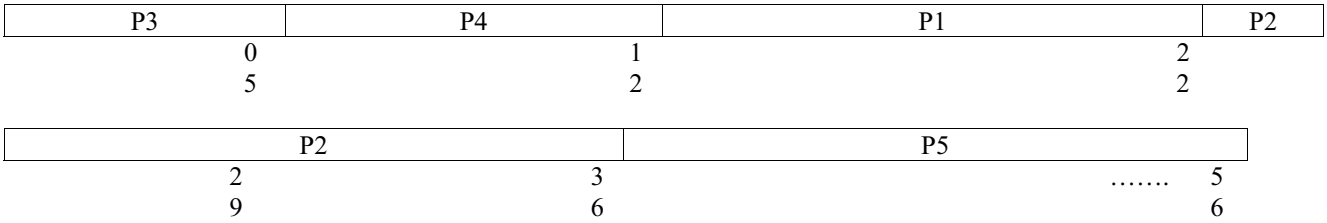
11. Processos I/O-bound são favorecidos neste tipo de escalonamento. Como a probabilidade desse tipo de processo sofrer preempção por tempo é baixa, a tendência é que os processos I/O-bound permaneçam nas filas de alta prioridade enquanto os processos CPU-bound tendem a posicionar-se nas filas de prioridade mais baixa.

12.

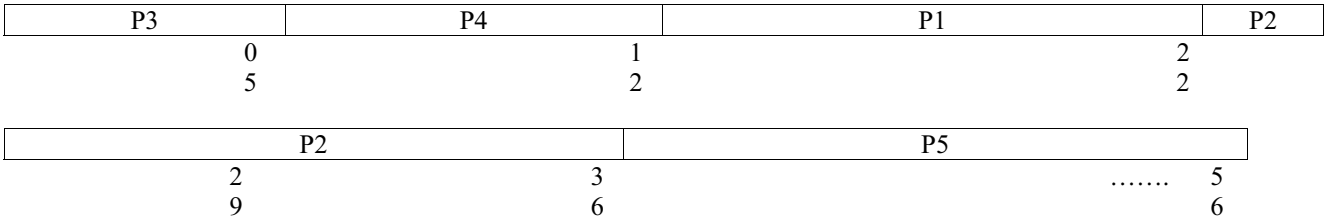
a)



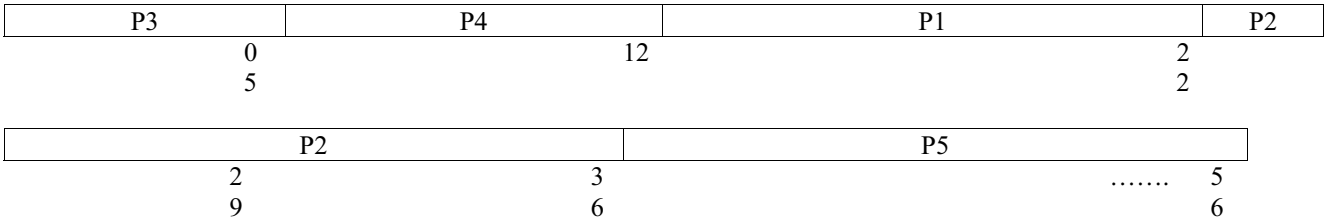
b)



c)



d)



13.

a) Instantes 24-25, 59-60

b) Nunca, pois o processo B possui maior prioridade do que o processo A.

c)

	00-04	05-09	10-14	15-19	20-24	25-29	30-34	35-39	40-44	45-49
ProcessoA	E	E	E	E	W	P	E	E	E	W
Processo B	P	P	P	P	E	E	W	W	P	E

	50-54	55-59	60-64	65-69	70-74	75-79	80-84	85-89	90-94	95-99	100-105	106- 115
Processo A	W	P	E	E	-	-	-	-	-	-	-	-
Processo B	E	E	W	W	W	P	E	E	W	W	P	E

14. Um valor de quantum grande pode prejudicar a multiprogramação, na medida em que a ocorrência de preempções por tempo é reduzida, favorecendo os processos CPU-bound e prejudicando os processos I/O-bound. Um valor de quantum pequeno ocasionaria um grande overhead ao sistema devido a alta frequência de mudanças de contexto geradas pelas frequentes preempções por tempo.

15.
a) P1: Execução, P2:Pronto, P3:Pronto
b) P1: Pronto, P2:Execução, P3:Pronto
c) P1: Terminado, P2:Terminado, P3:Execução

16.
a) P1: Espera, P2:Execução, P3:Pronto
b) P1: Pronto, P2:Terminado, P3:Execução
c) P1: Espera, P2:Terminado, P3:Terminado

17. A melhor política para minimizar o tempo de turnaround seria utilizar o escalonamento SJF na sequência de execução P3, P4, P2 e P1.

18.

a) FCFS

Com troca de contexto = 0 u.t.

P1		P2		P3		P4
0		0		1		
4		6		1		
0		0		0		

P4
1
4
0

Com troca de contexto = 5 u.t.

P1			P2			P3	
0	0		0	0			1
4	4		6	7			2
0	5		5	0			0

	P4	
1		1
2		5
5		5

b) SJF

Com troca de contexto = 0 u.t.

P2	P4	P1	P3
0	0	0	
2	5	9	
0	0	0	

P3
1
4
0

Com troca de contexto = 5 u.t.

P2		P4		P1		P3
0	0	0	0	1	1	
2	2	5	6	0	0	
0	5	5	0	0	5	

P3
1
5
5

c) Circular com fatia de tempo igual a 20 u.t.

Com troca de contexto = 0 u.t.

P1	P2	P3	P4	P1	P3
0	0	0	0	1	1
2	4	6	8	0	2
0	0	0	0	0	0

P4	P3
1	1
3	4
0	0

Com troca de contexto = 5 u.t.

P1		P2		P3		P4		P1
0	0	0	0	0	0	0	1	1
2	2	4	5	7	7	9	0	2
0	5	5	0	0	5	5	0	0

	P3		P4		P3
1	1	1	1	1	1
2	4	5	6	6	7
5	5	0	0	5	5

Capítulo 9 – Gerência de Memória

1. Maximizar o número de processos na memória, permitir a execução de programas maiores que a memória física, compartilhamento de dados na memória e proteção da memória utilizada por cada processo e pelo sistema operacional.

2. Considerando que o sistema opeacional e o programa somados ocupam ¾ da memória principal, temos 25% de subutilização da memória.

3. Como existe apenas 50Kb para a execução do programa, a memória deve ser dividida em duas áreas: uma para o módulo principal (20Kb) e outra de overlay para a carga dos módulos, em função do tamanho do maior módulo (30 Kb).

4. Não. No caso de não haver como aumentar o espaço de memória real, a única solução seria tentar alterar o programa de forma que o módulo de 40Kb pudesse ser dividido em outros módulos menores independentes.

5. Fragmentação interna ocorre em espaços livres e contíguos na memória principal que são pré-alocados por processos, não possibilitando, portanto, o uso por outros processos. Fragmentação externa ocorre em espaços livres e contínuos, porém tão pequenos que não possibilitam a alocação de programas por processos.

6.
a) 2Kb, 4Kb, 4Kb
b) 4Kb, 8Kb, 6Kb
c) não há fragmentação interna

7. Somente seria possível executar quatro programas concorrentemente alterando a configuração das partições do sistema e criando uma quarta partição. No segundo cado, seria possível executar um programa de 36Kb alterando a configuração do sistema, aumentando uma das partições e reduzindo as demais.

8. A grande diferença entre a alocação particionada estática absoluta e a alocação estática relocável é o local na memória principal onde programa é carregado. Na alocação absoluta, um programa pode apenas ser carregado a partir de um único endereço, consequentemente em uma única partição. Na alocação relocável, um programa pode ser carregado a partir de qualquer endereço ou partição.

9. No instante de tempo inicial, com a alocação dos processos por ordem crescente e alocação em múltiplos de 4Kb, a memória terá a seguinte disposição:

Sistema Operacional	20 Kb
Partição do Processo 1	32 Kb (30 Kb úteis)
Partição do Processo 2	8 Kb (6 Kb úteis)
Área Livre	4 Kb

Fragmentação interna na Partição do Processo 1: 2 Kb
Fragmentação interna na Partição do Processo 2: 2 Kb
Fragmentação externa: não há

No instante de tempo 5: O processo 1 termina sua execução.

Sistema Operacional	20 Kb
Área Livre	32 Kb
Partição do Processo 2	8 Kb (6 Kb úteis)
Área Livre	4 Kb

Fragmentação interna na Partição do Processo 2: 2 Kb
Fragmentação externa: 36 Kb

No instante de tempo 10: O processo 2 termina sua execução.

Sistema Operacional	20 Kb
Partição do Processo 3	36 Kb (36 Kb úteis)
Área Livre	8 Kb

Fragmentação interna na Partição do Processo 3: não há
Fragmentação externa: não há

10. Ver item 9.5.3.

11.

First-fit: 20Kb, 10Kb e 18Kb
Best-fit: 12Kb, 10Kb e 9Kb.
Worst-fit: 20Kb, 18Kb e 15Kb.

12.

Best-fit:

(a)

5 Kb	Programa A
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Livre
6 Kb	Programa C
26 Kb	Livre

(b)

5 Kb	Livre
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Livre
6 Kb	Programa C
26 Kb	Livre

(c)

5 Kb	Livre
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Program E
6 Kb	Programa C
26 Kb	Livre

Worst-fit:

(a)

5 Kb	Programa A
3 Kb	Programa B
10 Kb	Livre
6 Kb	Programa C
6 Kb	Programa D
20 Kb	Livre

(b)

5 Kb	Livre
3 Kb	Programa B
10 Kb	Livre
6 Kb	Programa C
6 Kb	Programa D
20 Kb	Livre

(c)

5 Kb	Livre
3 Kb	Programa B
10 Kb	Livre
6 Kb	Programa C
6 Kb	Programa D
4 Kb	Programa E
16 Kb	Livre

First-fit:

(a)

5 Kb	Programa A
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Livre
6 Kb	Programa C
26 Kb	Livre

(b)

5 Kb	Livre
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Livre
6 Kb	Programa C
26 Kb	Livre

(c)

4 Kb	Programa E
1 Kb	Livre
3 Kb	Programa B
6 Kb	Programa D
4 Kb	Livre
6 Kb	Programa C
26 Kb	Livre

13. A técnica de swapping foi introduzida para contornar o problema da insuficiência de memória principal. Essa técnica é aplicada à gerência de memória para programas que esperam por memória livre para serem executados. Nesta situação, o sistema escolhe um processo residente, que é transferido da memória principal para a memória secundária (swap out), geralmente disco. Posteriormente, o processo é carregado de volta da memória secundária para a memória principal (swap in) e pode continuar sua execução como se nada tivesse ocorrido.

14. O loader com relocação dinâmica permite que os programas possam ser retirados da memória principal para a memória secundária e trazidos novamente para a memória principal em qualquer posição.

Capítulo 10 – Gerência de Memória Virtual

1. Os principais benefícios da técnica de memória virtual são possibilitar que programas e dados sejam armazenados independente do tamanho da memória principal, permitir um número maior de processos compartilhando a memória principal e minimizar o problema da fragmentação. O que possibilita que um programa e seus dados ultrapassem os limites da memória principal é a técnica de gerência de memória virtual que combina as memórias principal e secundária, estendendo o espaço de endereçamento dos processos.

2. Ver item 10.4.

3. Porque caso o mapeamento fosse realizado para cada célula na memória principal, o espaço ocupado pelas tabelas de mapeamento seria tão grande quanto o espaço de endereçamento virtual de cada processo, o que inviabilizaria a implementação do mecanismo de memória virtual. Um processo em um sistema computacional com arquitetura de 32 bits poderia ter 4 G endereços virtuais e, consequentemente, tabelas de mapeamento com 4 G entradas.

4. A principal diferença entre os dois sistemas está relacionada a forma como o espaço de endereçamento virtual está dividido logicamente. Na paginação, o espaço de endereçamento está dividido em blocos com o mesmo número de endereços virtuais (páginas), enquanto que na segmentação o tamanho dos blocos pode variar (segmentos).

5. Página virtual é um conjunto de endereços virtuais que faz parte do espaço de endereçamento virtual de um processo. Página real é um conjunto de endereços reais localizado na memória principal. A página real está sempre associada a uma página virtual.

6. São tabelas de mapeamento, utilizadas no mecanismo de memória virtual, que possibilitam que endereços virtuais sejam traduzidos em endereços reais.

7. Para indicar se a página ou o segmento em questão encontra-se na memória principal.

8. O page fault ocorre todas as vezes que um processo faz referência a um endereço virtual pertencente a uma página virtual que não se encontra mapeada em uma página real, ou seja, não está, no momento, na memória principal. A ocorrência de um page fault é verificada através do bit de validade presente na ETP da tabela de páginas referente à página virtual. Uma elevada taxa de page fault pode comprometer o desempenho do sistema devido ao excessivo overhead de operações de E/S gerados pela paginação.

9. Não. A rotina de tratamento de page faults tem que permanecer sempre residente na memória principal, caso contrário não será possível realizar o page in quando necessário (no caso, até mesmo da própria rotina).

10. A fragmentação interna em um sistema que implementa paginação só é encontrada, realmente, na última página, quando o código não ocupa o frame por completo .

11. Na paginação por demanda, as páginas dos processos são transferidas da memória secundária para a principal apenas quando são referenciadas. Este mecanismo é conveniente, na medida em que leva para a memória principal apenas as páginas realmente necessárias à execução do programa. Desse modo, é possível que partes não executadas do programa, como rotinas de tratamento de erros, nunca sejam carregadas para a memória.

Na paginação antecipada, o sistema carrega para a memória principal, além das páginas referenciadas, outras páginas que podem ou não ser necessárias ao processo ao longo do seu processamento. Se imaginarmos que o programa está armazenado seqüencialmente no disco, existe uma grande economia de tempo em levar um conjunto de páginas da memória secundária, ao contrário de carregar uma de cada vez. Por outro lado, caso o processo não precise das páginas carregadas antecipadamente, o sistema terá perdido tempo e ocupado memória principal desnecessariamente.

12.A alocação fixa é simples de ser implementada pelo sistema operacional mas não é sempre uma boa opção pois os processos possuem necessidades diferentes na alocação de memória. A alocação variável é mais flexível mas exige que o sistema operacional monitore constantemente o comportamento dos processos gerando maior overhead.

13.
a)

NPV	Frame
9	4
10	9
34	3
65	7

b)

NPV	Frame
9	4
10	9
12	3
49	0
65	7

c) O endereço virtual possui 9 bits para endereçar a tabela de páginas e 9 bits para o deslocamento dentro da página.

d) O endereço virtual 4613 encontra-se na página virtual 9 (4613/512), que inicia no endereço virtual 4608. Como o deslocamento dentro do endereço virtual é 5, o endereço físico é a soma deste mesmo deslocamento ao endereço inicial do frame 2048, ou seja, 2053.

14.
a) Página 3 está mapeada no frame 10 que é o $11^{\underline{0}}$ frame da memória principal. Endereço físico: (11 x 2K)-1= 22.527
b) Página 2 está mapeada no frame 100 que é o $101^{\underline{0}}$ frame da memória principal. Endereço físico: (100 x 2K)= 204.800
c) Página 3 está mapeada no frame 10 que é o $11^{\underline{0}}$ frame da memória principal. O primeiro endereço da página 3 é (10x2K) = 20.480 somando ao deslocamento 10 = 20.490
d) 0, 1, 2, 3 e 6.

15.
a) Página virtual 1, deslocamento 51 e está na memória.
b) Página virtual 8, deslocamento 1 e não está na memória.
c) Página virtual 9, deslocamento 0 e está na memória.

16.
a) Pag. 2 (2048 / 3071), pag. 3 (3072 / 4095), pag. 5 (5120 / 6143) e pag. 7 (7168 / 8191).
b) End. Virtual 0 (PV 0 / Desloc 0) End. Real = 3072 + 0 = 3072
End. Virtual 1023 (PV 0 / Desloc 1023) End. Real = 3072 + 1023 = 4095
End. Virtual 1024 (PV 1 / Desloc 0) End. Real = 1024 + 0 = 1024
End. Virtual 6500 (PV 6 / Desloc 356) End. Real = 0 + 356 = 356
End. Virtual 3728 (PV 3 / Desloc 656) Page Fault

17. Ver item 10.4.3.

18. Para indicar se a página ou segmento foi modificado desde o momento em que foi carregado pela útlima vez na memória principal.

19. O princípio da localidade é fundamental para qualquer sistema que implemente a gerência de memória virtual, pois reduz a ocorrência de page/segments faults e, conseqüentemente, operações de E/S.

20. Porque o princípio da localidade não se faz presente em códigos desestrurados.

21. Ver item 10.4.5.

22.

a) 11 bits.

b) $2^{24}/2^{11} = 2^{13}$

c) 10.240+2.047=12.287

d) (00080A)16 = 2058, página virtual 1, deslocamento 100e endereço físico igual a 100.

e) Páginas 0 e 2 pois estão na memória principal e possuem BM=1.

23.

a) 1 / 2 / 3 / 1 / 4 / 2 / 5 / 3 / 4 / 3

FIFO = Total PF = 5 (melhor política)

1	2	3	1	4	2	5	3	4	3
PF	PF	PF	-	PF (sai 1)	-	PF (sai 2)	-	-	-

LRU = Total PF = 8

1	2	3	1	4	2	5	3	4	3
PF	PF	PF	-	PF (sai 2)	PF (sai 3)	PF (sai 1)	PF (sai 4)	PF (sai 2)	-

b) 1 / 2 / 3 / 1 / 4 / 1 / 3 / 2 / 3 / 3

FIFO = Total PF = 7

1	2	3	1	4	1	3	2	3	3
PF	PF	PF	-	PF (sai 1)	PF (sai 2)	-	PF (sai 3)	PF (sai 4)	-

LRU = Total PF = 5 (melhor política)

1	2	3	1	4	1	3	2	3	3
PF	PF	PF	-	PF (sai 2)	-	-	PF (sai 4)	-	-

24.

a) 4 bits para o número da página, possibilitando endereçar as 16 páginas possíveis, e 12 bits para o deslocamento, possibilitando endereçar os 4K endereços de uma página.

b)

Página Referenciada	Página Removida	Page Fault (sim/não)
5	-	Não
15	0	Sim
12	2	Sim
8	1	Sim
0	9	Sim

25.
a) 5 bits para o número da página e 11 bits para o deslocamento.

b)

Página virtual	Páginas na memória	Page fault	Página a ser substituida
0	-	S	-
7	0	S	-
2	7, 0	S	-
7	2, 7, 0	N	-
5	7, 2, 0	S	-
8	5, 7, 2, 0	S	0
9	8, 5, 7, 2	S	2
2	9, 8, 5, 7	S	7
4	2, 9, 8, 5	S	5
-	4, 2, 9, 8	-	-

26.
a) Nos instantes (T + 3) quando a página 3 é descartada e (T + 4) quando a página 6 é descartada. Ambas as páginas têm indicativo que sofreram modificação, sendo necessário armazená-las no arquivo de paginação.
b) No instante (T + 1) pois com o page in da página 9 chega-se ao limite de 10 páginas na memória principal.
c) A página 3 já que é a primeira a ser descartada.
d) Através do bit de validade da página 1.

27.
a) Frame 0.
b) Frame 2.
c) Frame 1.

28.

FIFO gera 6 page faults

Página virtual	Páginas na memória	Page fault	Página a ser substituida
0	-	S	-
1	0	S	
7	1, 0	S	
2	7, 1, 0	S	
3	2, 7, 1, 0	S	0
2	3, 2, 7, 1	N	
7	3, 2, 7, 1	N	
1	3, 2, 7, 1	N	
0	3, 2, 7, 1	S	1
3	0, 3, 2, 7	N	
-	0, 3, 2, 7	-	-

LRU gera 7 page faults

Página virtual	Páginas na memória	Page fault	Página a ser substituida
0	-	S	-
1	0	S	
7	1, 0	S	
2	7, 1, 0	S	
3	2, 7, 1, 0	S	0
2	3, 2, 7, 1	N	
7	2, 3, 7, 1	N	
1	7, 2, 3, 1	N	
0	1, 7, 2, 3	S	3
3	0, 1, 7, 2	S	2
-	3, 0, 1, 7	-	-

29. Um buffer de páginas modificadas permite adiar a gravação de páginas modificadas que foram selecionadas para realocação e seriam gravadas em disco, otimizando o desempenho do sistema.

30. Existe uma relação entre o tamanho da página e o número de operações de E/S que o sistema deverá executar para carregar as páginas da memória secundária para a memória principal. Quanto menor o tamanho da página, maior o número de operações de E/S, aumentando a taxa de paginação. Por outro lado, páginas pequenas oferecem menor fragmentação interna.

31. O endereço virtual é formado por NPV1 com 2 bits, NPV2 com 21 bits e deslocamento com 9 bits.

32. Como existem 12 bits para o deslocamento, temos 2^{12} endereços, ou seja, páginas de 4Kb. Como existem 20 bits para o endereçamento de páginas virtuais, temos 2^{20} páginas possíveis.

33. O endereço virtual é formado por NPV1 com 8 bits, NPV2 com 6 bits, NPV3 com 6 bits e deslocamento com 12 bits. Se um processo, quando fosse criado, alocasse todas as tabelas de páginas possíveis, o processo teria tabelas que somadas endereçariam 220 posições. Para evitar que a memória principal seja ocupada com páginas que talvez nunca sejam utilizadas, o processo aloca suas tabelas de páginas dinamicamente, conforme a necessidade. Por exemplo, quando o processo for criado, apenas uma página de nível 1 (28 posições), uma página de nível 2 (26 posições) e uma página de nível 3 (26 posições) são alocadas inicialmente.

34. O tempo de acesso à memória (TAM) é calculado a partir das taxas de acerto e falha na TLB, ou seja, TAM é igual a (0.98% * tempo de acesso à TLB) + (número de níveis * (0.02% * tempo de acesso à memória)) = (0.98%*0) + (2 * (0.02%*100)) = 4ns. O tempo total para a execução da instrução é igual a soma do tempo de leitura da instrução (4ns), do tempo de leitura do operando "direto" (4ns) e do tempo de leitura do operando "indireto" (8ns), totalizando 16 ns.

35. Ver item 10.6.

36. Na maioria das políticas, o critério de escolha considera o estado do processo e sua prioridade, buscando dessa forma identificar o processo com as menores chances de serem executados.

37. O problema da fragmentação existe tanto na gerência de memória virtual por paginação quanto na por segmentação. A fragmentação interna ocorre na memória virtual por paginação na última página, caso não seja totalmente ocupada. A fragmentação externa ocorre na memória virtual por segmentação em função dos espaços livres deixados entre segmentos alocados na memória principal.

38. Thrashing é conseqüência da excessiva paginação/segmentação em sistemas que implementam memória virtual, levando o sistema a dedicar mais tempo com operações relacionadas à gerência da memória do que no processamento das aplicações dos usuários.

Capítulo 11 – Sistema de Arquivos

1. Um arquivo é um conjunto de registros definidos pelo sistema de arquivos, tornando seu conceito abstrato e generalista Um arquivo é constituído por informações logicamente relacionadas, podendo representar instruções ou dados. Arquivos são gerenciados pelo sistema operacional de maneira a facilitar o acesso dos usuários ao seu conteúdo.

2. A forma mais simples de organização de arquivos é através de uma seqüência não-estruturada de bytes, na qual o sistema de arquivos não impõe nenhuma estrutura lógica para os dados. Alguns sistemas operacionais possuem diferentes organizações de arquivos. Neste caso, cada arquivo criado deve seguir um modelo suportado pelo sistema de arquivos. As organizações mais conhecidas e implementadas são a seqüencial, relativa e indexada

3. No método de acesso seqüencial, a leitura dos registros é realizada na ordem em que são gravados e a gravação de novos registros só é possível no final do arquivo. No acesso direto, a leitura/gravação de um registro ocorre diretamente na sua posição, através do número do registro que é a sua posição relativa ao início do arquivo. No acesso indexado, o arquivo possui uma área de índice onde existem ponteiros para os diversos registros. Sempre que a aplicação deseja acessar um registro, deve ser especificada uma chave através da qual o sistema pesquisará na área de índice o ponteiro correspondente

4. Possibilitar o acesso as rotinas de E/S que têm como função disponibilizar uma interface simples e uniforme entre a aplicação e os diversos dipositivos.

5. Estrutura de diretório de nível único, com dois níveis e em árvore. A especificação de cada estrutura esta em 11.3.

6. Ver item 11.4.

7. A alocação contígua consiste em armazenar um arquivo em blocos seqüencialmente dispostos no disco. A desfragmentação pode solucionar o problema da fragmentação reorganizando todos os arquivos no disco de maneira que só exista um único segmento de blocos livres.

8. Ver itens 11.5.2 e 11.5.3.

9. Senha de acesso, proteção por grupos de usuários e lista de controle de acesso. A vantagem da associação de uma senha de acesso a um arquivo é a simplicidade, pois o controle resume-se ao usuário ter conhecimento da senha e, conseqüentemente, ter a liberação do acesso ao arquivo concedida pelo sistema. A vantagem da proteção por grupos de usuários é oferecer uma proteção em três níveis: owner (dono), group (grupo) e all (todos). Já a lista de controle de acesso tem a vantagem de especifar individualmente para cada arquivo qual usuário e tipo de acesso é concedido.

10. É a técnica em que o sistema operacional reserva uma área da memória para que se tornem disponíveis caches utilizados em operações de acesso ao disco. Quando uma operação é realizada, o sistema verifica se a informação desejada se encontra no buffer cache. Em caso positivo, não é necessário o acesso ao disco. Caso o bloco requisitado não se encontre no cache, a operação de E/S é realizada e o cache é atualizado.

Capítulo 12 – Gerência de Dispositivos

1. A gerência de dispositivos é estruturada através de camadas em um modelo semelhante ao apresentado para o sistema operacional como um todo. As camadas de mais baixo nível escondem características dos dispositivos das camadas superiores, oferecendo uma interface simples e confiável ao usuário e suas aplicações. As camadas são divididas em dois grupos, onde o primeiro grupo visualiza os diversos tipos de dispositivos do sistema de um modo único, enquanto o segundo é específico para cada dispositivo. A maior parte das camadas trabalha de forma independente do dispositivo.

2. Tornar as operações de E/S o mais simples possível para o usuário e suas aplicações. Com isso, é possível ao usuário realizar operações de E/S sem se preocupar com detalhes do dispositivo que está sendo acessado.

3. Por comandos de leitura/gravação e chamadas a bibliotecas de rotinas oferecidas por linguagens de alto nível ou diretamente através de uma system call em um código de alto nível.

4. Criar uma interface padronizada com os device drivers e oferecer uma interface uniforme com as camadas superiores.

5. Implementar a comunicação do subsistema de E/S com os dispositivos, através de controladores.

6. Para que seja possível a inclusão de novos drivers sem a necessidade de alteração da camada de subsistema de E/S.

7. De forma simplificada, uma operação de leitura em disco utilizando DMA teria os seguintes passos. A UCP, através do device driver, inicializa os registradores do controlador de DMA e, a partir deste ponto, fica livre para realizar outras atividades. O controlador de DMA, por sua vez, solicita ao controlador de disco a transferência do bloco do disco para o seu buffer interno. Terminada a transferência, o controlador de disco verifica a existência de erros e, caso não haja erros, o controlador de DMA transfere o bloco para o buffer de E/S na memória principal. Ao término da transferência, o controlador de DMA gera uma interrupção avisando ao processador que o dado já encontra-se na memória principal. A principal vantagem dessa técnica é evitar que o processador fique ocupado com a transferência do bloco para a memória.

8. Os dispositivos estruturados (block devices) caracterizam-se por armazenar informações em blocos de tamanho fixo, possuindo cada qual um endereço que podem ser lidos ou gravados de forma independente dos demais. Discos magnéticos e ópticos são exemplos de dispositivos estruturados. Os dispositivos não-estruturados são aqueles que enviam ou recebem uma seqüência de caracteres sem estar estruturada no formato de um bloco. Desse modo, a seqüência de caracteres não é endereçável, não permitindo operações de acesso direto ao dado. Dispositivos como terminais, impressoras e interfaces de rede são exemplos de dispositivos não-estruturados.

9. A principal razão é o aspecto mecânico presente nas arquiteturas de fitas e discos magnéticos, devido a isso, o tempo total das operações de E/S é extremamente longo, se comparado ao número de instruções que o processador pode executar no mesmo intervalo de tempo.

10. São técnicas que possibilitam garantir a integridade dos dados mesmo en caso de crash nos discos magnéticos.

11. Ver item 12.7.1.

Capítulo 13 – Sistemas com Múltiplos Processadores

1. A maior limitação é que somente uma unidade funcional pode estar utilizando o barramento em determinado instante, o que pode produzir um gargalo quando várias unidades tentam acessá-lo simultaneamente. Além disso, caso ocorra algum problema no barramento, todo o sistema ficará comprometido.

2. O esquema de cache é utilizado para reduzir a latência das operações de acesso à memória principal.

3. No mecanismo write-through, sempre que um dado no cache for alterado, a memória principal também seja atualizada. O mecanismo write-back permite alterar o dado no cache e não alterá-lo imediatamente na memória principal, oferecendo, assim, melhor desempenho.

4. Na técnica write-invalidate, como o nome sugere, sempre que ocorre uma operação de escrita, todas as cópias do mesmo dado em outros caches são invalidadas e o novo valor atualizado na memória principal. Nesse caso, apenas o processador que realizou a operação de escrita possuirá uma cópia válida do dado no cache. Na técnica write-update, sempre que ocorre uma operação de escrita, a alteração é informada pelo barramento para que todos os processadores que possuam a mesma cópia atualizem seu cache com o novo valor.

5. Em uma arquitetura de barramento cruzado seriam necessários 64 processadores enquanto que em uma rede Ômega apenas 48.

6. 2*(n-1)=30 onde n=15 (número de comutadores – 1)

7. A diferença entre os sistemas NUMA e SMP está no desempenho das operações de acesso à memória. Em uma arquitetura NUMA, onde cada conjunto possui três processadores e uma memória principal conectados a um mesmo barramento interno, os conjuntos são interligados através de um barramento inter-conjunto, criando um modelo hierárquico. Quando um processador faz um acesso local, ou seja, à memória dentro do mesmo conjunto, o tempo de acesso é muito menor que um acesso à memória remota em um outro conjunto. Para manter um nível de desempenho satisfatório, o sistema deve manter a maioria dos acessos locais, evitando acessos à memória remota.

8. A topologia dos sistemas SMP limita o número máximo de UCPs da configuração, pois o acesso à memória pelos processadores ocorre em um tempo sempre uniforme. Os sistemas NUMA permitem reunir vários processadores em conjuntos e interligar estes conjuntos através de diversas topologias. Como conseqüência, o tempo de acesso à memória não é mais uniforme.

9. Clusters são sistemas fracamente acoplados, formados por nós que são conectados por uma rede de interconexão de alto desempenho dedicada. Cada nó da rede é denominado membro do cluster, possuindo seus próprios recursos, como processadores, memória, dispositivos de E/S e sistema operacional. Geralmente, os membros do cluster são de um mesmo fabricante, principalmente por questões de incompatibilidade dos sistemas operacionais. A razão para o surgimento e a rápida aceitação de sistemas em cluster foi a maior necessidade de tolerância a falhas e alta disponibilidade, de forma reduzir o downtime.

10. O conceito de imagem única do sistema está presente em sistemas distribuídos, que é um sistema fracamente acoplado pelo aspecto de hardware e fortemente acoplado pelo aspecto de software. Para o usuário e suas aplicações, é como se não existisse uma rede de computadores independente, mas sim um único sistema fortemente acoplado.

11. Em sistemas distribuídos, o conceito de transparência torna-se fator chave, pois, a partir dele, um conjunto de sistemas independentes parece ser um sistema único, criando a idéia da imagem única do sistema.

12.

a) Circular com fatia de tempo igual a 5

CPU 1	
CPU 2	

Tempo		**P1**	**P2**	**P3**	**P4**	**P5**
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						

Tempo de turnaround:

P1: 12 ut
P2: 11 ut
P3: 18 ut
P4: 18 ut
P5: 13 ut

b) Prioridade (número menor implica prioridade maior)

CPU 1	
CPU 2	

Tempo		**P1**	**P2**	**P3**	**P4**	**P5**
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						

Tempo de turnaround:

P1: 10 ut
P2: 20 ut
P3: 22 ut
P4: 11 ut
P5: 16 ut