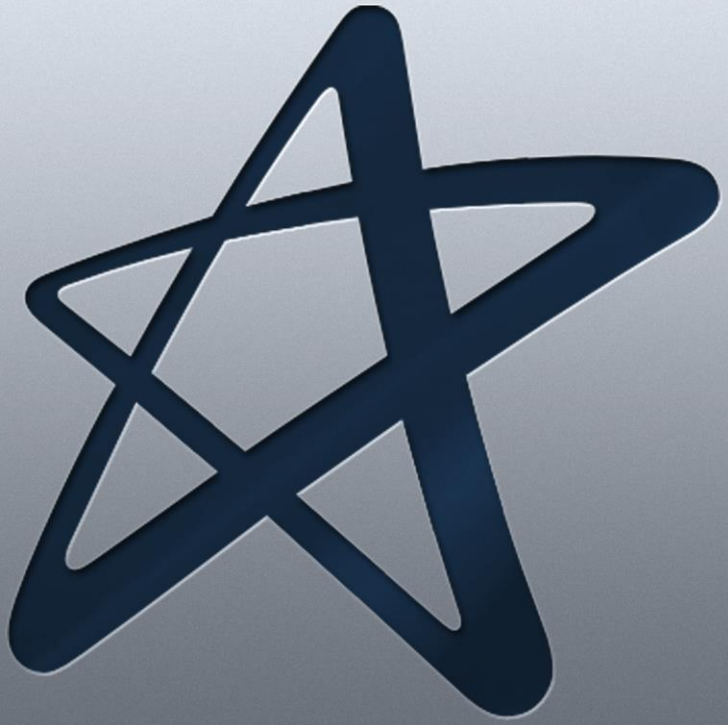
Tópicos da Informática



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material teórico

Sistemas Operacionais

Responsável pelo Conteúdo:
Profª. Dr. Luciano Silva

Revisão Técnica:
Prof. Ms. Fábio Peppe Beraldo

Revisão Textual:
Profª. Esp. Márcia Ota

UNIDADE Sistemas Operacionais

Caros alunos,
Seja bem-vindos à nossa Unidade de Conteúdo II!

Aqui, nosso objetivo será estudar com maiores detalhes os principais componentes de um sistema operacional: gerenciamento de processos, de memória, de arquivos e de dispositivos.

Atenção

Para um bom aproveitamento do curso, leia o material teórico atentamente antes de realizar as atividades. É importante também respeitar os prazos estabelecidos no cronograma.

Contextualização

Nosso objetivo, nesta terceira unidade de conteúdo, será estudar, de um ponto de vista mais aprofundado, a organização básica do núcleo de um sistema operacional.

O estudo dos principais componentes do núcleo de um sistema operacional (gerenciador de processos, de memória, de arquivos e de dispositivos) é importante para qualquer desenvolvimento em Computação. Quando você, por exemplo, carrega um arquivo de um trabalho elaborado no Microsoft Word, esses quatro gerenciadores trabalham em conjunto para realizar a tarefa:

- O Microsoft Word "executará" como um processo dentro do sistema operacional Windows e será controlado pelo gerenciador de processos.
- Para que o programa Microsoft Word possa ser carregado na memória, é necessário o trabalho do gerenciador de memória.
- A localização do arquivo aberto no disco é de responsabilidade do gerenciador de arquivos, enquanto que o acesso ao disco é realizado pelo gerenciador de dispositivos.

Material Teórico

1. Introdução

O objetivo desta unidade de conteúdo será estudar, de forma mais detalhada, a estrutura de um segundo ambiente: os sistemas operacionais. Daremos particular atenção aos quatro elementos que compõem a estrutura dos modernos sistemas operacionais:

- O gerenciador de processos.
- O gerenciador de memória.
- O gerenciador de arquivos.
- O gerenciador de dispositivos.

Para que você possa entender esses quatro conceitos, esta unidade está organizada da seguinte forma:

- Na *Seção 2,* você revisará os sistemas operacionais que foram apresentados na primeira unidade desse curso.
- A *Seção 3* lhe detalhará o gerenciador de processos.
- A *Seção 4* irá lhe apresentar o gerenciador de memória.
- Na *Seção 5*, você saberá mais sobre o gerenciador de arquivos.
- E, finalmente, na *Seção 6* lhe será mostrado o gerenciador de dispositivos.

Ao final das atividades desta unidade, faça uma autoavaliação, considerando se é capaz de:

- Entender e caracterizar os quatro principais componentes de um sistema operacional (gerenciador de processos, gerenciador de memória, gerenciador de arquivos e gerenciador de dispositivos).
- Compreender seus principais termos.
- Perceber como esses componentes se interligam.

Para refletir um pouco mais sobre questões relacionadas ao conteúdo desta unidade, acesse, no ambiente *online*, os fóruns dessa unidade para apresentar e discutir qualquer dificuldade encontrada.

Bom estudo!

2. Revisão de sistemas operacionais

Do ponto de vista dos ambientes operacionais, a categoria dos sistemas operacionais é uma das mais importantes entre os programas existentes. Conforme já apresentado anteriormente, um sistema operacional funciona como mediador entre diversos programas e os sistemas de *hardware*. Esse tipo de mediador é formado por um conjunto de rotinas que oferecem serviços essenciais aos usuários, às suas aplicações e também ao próprio sistema operacional. A esse conjunto de rotinas dá-se o nome de núcleo do sistema ou *kernel*, conforme ilustra a Figura 1:

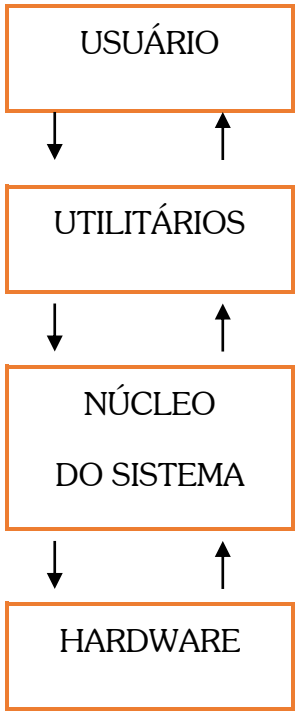


Figura 1: Hierarquia de um sistema operacional.

É fundamental não confundir o núcleo do sistema com aplicações, utilitários ou com o interpretador de comandos, que acompanham o sistema operacional, pois essas aplicações são utilizadas de maneira transparente pelos usuários, escondendo todos os detalhes da interação com o sistema. Por sua vez, os utilitários, como os compiladores, editores de texto e interpretadores de comandos, permitem aos usuários, desenvolvedores e administradores de sistema uma interação amigável com o sistema operacional.

Existe, contudo, grande dificuldade em compreender a estrutura e o funcionamento do sistema operacional, afinal, esse não é executado como uma aplicação tipicamente sequencial: com início, meio e fim.

Diferente dessa lógica linear, os procedimentos do sistema operacional são executados concorrentemente, ou seja, sem uma ordem específica ou predefinida, com base em eventos

dissociados do tempo. Muitos desses eventos estão relacionados ao *hardware* e às tarefas internas do próprio sistema operacional.

Do ponto de vista estrutural, o núcleo de um sistema operacional é formado por quatro módulos básicos, tal qual mostra a Figura 2:

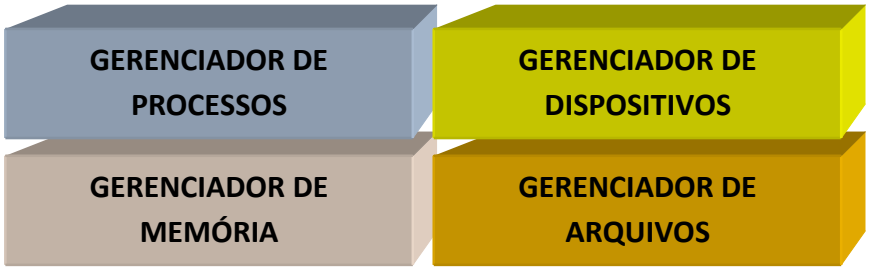


Figura 2: Módulos de um sistema operacional.

Cada programa executado em um sistema operacional é chamado de processo. Sendo assim, o módulo capaz de gerenciar a execução desses programas é denominado *gerenciador de processos.*

Para que seja possível armazenar os programas em memória, é necessário decidir como carregá-los e qual a porção de memória alocada para cada programa. Essas decisões são tomadas pelo *gerenciador de memória.*

O controle dos dispositivos de Entrada e Saída (E/S) é efetuado pelo *gerenciador de dispositivos* que, entre outras tarefas, é responsável pelo acesso aos *device drivers*, programas que permitem a comunicação entre os dispositivos. Para dispositivos com grande capacidade de armazenamento, por exemplo, um *Hard Disk* (HD), são necessários de meios mais eficientes de acesso aos dados armazenados. Nesse caso, é comum se utilizar uma estrutura de arquivos e diretórios para aumentar a eficiência desse acesso. Essas estruturas são controladas pelo *gerenciador de arquivos.*

3. Gerenciamento de processos

O conceito de processo é a base para a implementação de um gerenciador de processo. O processador é projetado apenas para executar instruções, não sendo capaz de distinguir qual programa se encontra em execução. A gerência de um ambiente multiprogramável é função exclusiva do sistema operacional, que deve controlar a execução dos diversos programas e o uso concorrente do processador.

A gerência do processador é uma das principais funções de um sistema operacional. É por meio dos processos que um programa pode alocar recursos, compartilhar dados e trocar informações.

Um processo pode ser entendido, inicialmente, como um programa em execução, que tem suas informações mantidas pelo sistema operacional. Em um sistema multiusuário, cada usuário tem a impressão de possuir o processador e todos os demais recursos reservados exclusivamente para si, mas isso não acontece de fato. Nesse caso, todos os recursos estão compartilhados, inclusive a CPU, onde o processador executa o processo desse usuário por um intervalo de tempo e, no instante seguinte, poderá processar outro programa, do mesmo ou de outro usuário.

Para essa troca de processos ocorrer sem problemas, é necessário que todas as informações do programa interrompido sejam guardadas, a fim de que esse possa retornar à CPU exatamente do ponto em que parou, sem faltar nenhuma informação vital à sua continuação. Assim, todas as informações necessárias à execução de determinado programa fazem parte do processo.

Um processo também pode ser definido como o ambiente, onde o programa é executado. Além das informações sobre a execução, esse ambiente também possui a quantidade suficiente de recursos do sistema que o programa necessita utilizar, como espaço de endereçamento, tempo do processador e área em disco.

Um processo é formado por três partes: contexto de *software*, contexto de *hardware* e espaço de endereçamento, que juntas mantêm todas informações necessárias à execução de um programa. Esses termos são definidos por:

- *Contexto de software:* nesse contexto, são especificadas as características e limites dos recursos que podem ser alocados pelo processo, como número máximo de arquivos abertos; prioridade de execução; número máximo de linhas impressas, entre outros aspectos. Muitas dessas características são criadas no momento da execução do processo, exatamente em sua alocação.

- *Contexto de hardware:* armazena o conteúdo dos registradores gerais da CPU, além dos registradores de uso específico. Quando um processo está em execução, seu contexto de *hardware* está armazenado nos registradores da CPU. No momento em que o processo perde a utilização da CPU, o sistema salva as informações no contexto de *hardware* do processo. A troca de um processo por outro no processador, comandada pelo sistema operacional, é denominada *troca de contexto*, que consiste em salvar o conteúdo dos registradores do processo que está deixando a CPU e carregá-los com os valores referentes ao do novo processo que executará. Essa operação resume-se em substituir o contexto de *hardware* de um processo pelo de outro.

- *Espaço de endereçamento:* área de memória pertencente a um processo, onde as instruções e os dados do programa são armazenados para execução. Cada processo possui seu próprio espaço de endereçamento, que deve ter seu acesso devidamente protegido dos demais processos. Os contextos de *software* e de *hardware* não fazem parte do espaço de endereçamento.

A partir do momento em que vários processos estiverem no estado de pronto, devem ser estabelecidos critérios para definir qual processo será escolhido para fazer uso do processador. Tais critérios compõem a política de escalonamento, que é a base da gerência do processador e da multiprogramação em um sistema operacional. Entre as funções da gerência do processador, podemos citar:

- Manter o processador ocupado a maior parte do tempo.
- Balancear o uso da CPU entre processos.
- Privilegiar a execução de aplicações críticas.
- Maximizar o *throughput.*
- Oferecer razoáveis tempos de resposta aos usuários interativos.

Cada sistema operacional possui sua política de escalonamento adequada ao seu propósito e às suas características. Sistemas de tempo compartilhado, por exemplo, possuem requisitos de escalonamento distintos dos sistemas de tempo real. Os critérios utilizados em uma estratégia de escalonamento são os seguintes:

- *Utilização do processador:* corresponde a uma taxa de utilização, que na maioria dos sistemas varia entre 30 e 90%. Uma utilização abaixo dos 30% indica um sistema ocioso, com carga de processamento baixa, enquanto uma taxa de utilização acima dos 90% pode apontar um sistema bastante carregado, próximo de sua capacidade máxima (em alguns casos tal situação pode levar a um *crash* – travamento do sistema).
- *Throughput:* corresponde ao número de processos executados em um determinado intervalo de tempo. Quanto maior o *throughput*, maior será o número de tarefas executadas em função do tempo. A maximização do *throughput* é desejada na maioria dos sistemas.
- *Tempo de processador:* é o tempo que um processo leva no estado de execução, durante seu processamento. As políticas de escalonamento não interferem nesse parâmetro, sendo esse tempo função apenas do código executável e da E/S de dados.
- *Tempo de espera (pela CPU):* diz respeito a todo o tempo que o processo permanece na fila de pronto, aguardando a liberação da CPU para executar. A redução desse tempo de espera é desejada pela maioria das políticas de escalonamento.
- *Tempo de Turnaround:* trata-se do tempo total que o processo permaneceu no sistema, desde sua criação até o momento em que foi encerrado. São contados os tempos de alocação de memória, espera na fila de pronto e interrupção (E/S).
- *Tempo de Resposta:* é o tempo decorrido entre uma requisição ao sistema e o instante em que a resposta começa a ser exibida. Em sistemas interativos, como aplicações *online* ou acesso à *Web*, os tempos de resposta devem ser da ordem de apenas poucos segundos.

A seguir, serão apresentados alguns tipos de escalonamentos encontrados nos modernos sistemas operacionais. Escalonamentos do tipo não-preemptivos são aqueles onde o sistema operacional não pode interromper o processo em execução para retirá-lo da CPU. Assim sendo, se nenhum evento externo ocorresse durante a execução do processo, esse permaneceria na CPU até seu término, ou até que alguma instrução do próprio programa o desviasse para o estado de espera (operação de E/S).

São escalonamentos não-preemptivos:

- *FIFO*: onde o processo que chegar primeiro à fila de pronto é selecionado para execução, utilizando o processador até terminar sua execução ou ser interrompido por E/S. Neste caso, o próximo processo da fila de pronto é selecionado para execução. Todo processo que chega à fila de pronto entra no final dessa sequência, conservando sua ordem de chegada à fila, até ser escalonado novamente. Apesar de simples, esse escalonamento apresenta algumas deficiências, principalmente no que diz respeito à dificuldade de se prever o início da execução de um processo, já que a ordem de chegada à fila de pronto deve ser observada à risca. Outro problema é quanto aos tipos de processo, onde os *CPU-bound* levam vantagem no uso do processador em relação aos *I/O-bound*, pois o sistema não reconhece esse tipo de diferença. O escalonamento *FIFO* foi, inicialmente, implementado em sistemas monoprogramáveis, tornando-se ineficiente se aplicado em sistemas interativos de tempo compartilhado.

- *SJF (Shortest Job First)*: esse escalonamento seleciona o processo que tiver o menor tempo de processador por executar. Dessa forma, o processo que estiver na fila de pronto com menor necessidade de tempo de CPU para terminar o seu processamento será o escolhido para ocupá-la. Funciona com um parâmetro passado ao sistema via contexto de *software*, onde o tempo estimado para o processo é informado baseando-se em estatísticas de execuções anteriores.

- *Cooperativo*: esse escalonamento busca aumentar o grau de concorrência no processador. Nesse caso, um processo em execução pode, voluntariamente, liberar o processador, retornando-o à fila de pronto, possibilitando que um novo processo seja escalonado, o que permite melhor distribuição do tempo do processador. A liberação da CPU é uma tarefa exclusiva do programa em execução, que de maneira cooperativa libera o processador para outro processo. Nesse mecanismo, o processo em execução verifica periodicamente uma sequência de mensagens para saber se existem outros processos na fila de pronto. Porém, como a interrupção do processo não depende do sistema operacional, situações indesejáveis podem ocorrer. Por exemplo: se um programa em execução não verificar a fila de mensagens, os demais programas não terão chance de executar enquanto a CPU não for liberada. As primeiras versões do *Windows* chegaram a utilizar esse tipo de escalonamento.

Já os escalonamentos preemptivos são caracterizados pela possibilidade de o sistema operacional interromper o processo em execução para retirá-lo da CPU e dar lugar a outro. Nesse caso, o processo retirado da CPU volta ao estado de pronto, onde permanece aguardando nova oportunidade de ocupá-la. Com o uso da preempção, é possível ao sistema priorizar a execução de processos, como no caso de aplicações em tempo real. Outro

benefício é a possibilidade de implementar políticas de escalonamento que compartilhem o processador de uma maneira mais uniforme, balanceando o uso da CPU entre os processos.

São escalonamentos preemptivos:

- *Circular*: é um tipo de escalonamento projetado especialmente para sistemas em tempo compartilhado. É muito semelhante ao *FIFO* (obedece a ordem de chegada à fila de pronto), mas quando um processo passa para o estado de execução, há um limite de tempo para o uso contínuo do processador, chamado *fatia de tempo* (*time-slice*) ou *quantum*. Assim, toda vez que um processo é selecionado para execução uma nova fatia de tempo lhe é concedida. Caso essa fatia de tempo expire, o sistema operacional interrompe o processo, salva seu contexto e o direciona para a fila de pronto. Esse mecanismo é conhecido como *preempção por tempo*. A principal vantagem desse escalonamento é não permitir que um processo monopolize a CPU. Por outro lado, uma desvantagem é que os processos *CPU-bound* são beneficiados no uso do processador em relação aos processos *I/O-bound*, pois tendem a utilizar totalmente a fatia de tempo recebida.

- *Por prioridades*: funciona com base em um valor associado a cada processo, denominado *prioridade de execução*. O processo com maior prioridade na fila de pronto sempre é o escolhido para ocupar o processador. Os processos com prioridades equivalentes são escalonados pelo critério *FIFO*. Nesse escalonamento, o conceito da fatia de tempo não existe. Como consequência, um processo em execução não pode sofrer preempção por tempo. Aqui, a perda do uso do processador somente ocorrerá no caso de uma mudança voluntária para o estado de espera (interrupção por E/S), ou quando outro processo de prioridade maior passa (ou chega) para o estado de pronto. Nesse caso, o sistema operacional interrompe o processo em execução, salva seu contexto e o coloca na fila de pronto, dando lugar na CPU ao processo prioritário. Esse mecanismo é chamado de *preempção por prioridade*;.

- *Escalonamento circular com prioridades*: implementa o conceito de fatia de tempo e de prioridade de execução associada a cada processo. Nesse escalonamento, um processo permanece no estado de execução até que termine seu processamento ou, voluntariamente, passe para o estado de espera (interrupção por E/S), ou ainda sofra uma preempção por tempo ou prioridade. A principal vantagem desse escalonamento é permitir um melhor balanceamento no uso do processador, com a possibilidade de diferenciar o grau de importância dos processos por meio da prioridade. O *Windows* utiliza esse tipo de escalonamento.

- *Por múltiplas filas*: esse escalonamento implementa várias filas de pronto, cada uma com prioridade específica. Os processos são associados às filas de acordo com características próprias, como importância da aplicação, tipo de processamento ou área de memória necessária. Assim, não é o processo que detém a prioridade, mas sim a fila. O processo em execução sofre preempção, caso outro processo entre em uma fila de maior prioridade. O sistema operacional só pode escalonar processos de uma fila quando todas as outras filas de maior prioridade estiverem vazias. Aqui, os processos sempre voltam para a mesma fila de onde saíram.

- *Por múltiplas filas com realimentação*: esse é semelhante ao anterior, porém permitindo ao processo voltar para outra fila de maior ou menor prioridade, de acordo com seu

comportamento durante o processamento. O sistema operacional identifica dinamicamente o comportamento de cada processo e o redireciona para a fila mais conveniente ao longo de seu processamento. É um algoritmo generalista, que pode ser implementado na maioria dos sistemas operacionais.

4. Gerenciamento de memória

Historicamente, a memória principal sempre foi vista como um recurso escasso e caro. Uma das maiores preocupações dos projetistas foi desenvolver sistemas operacionais que não ocupassem muito espaço de memória e, ao mesmo tempo, otimizassem a utilização dos recursos computacionais. Mesmo atualmente, com a redução do custo e o considerável aumento da capacidade da memória principal, seu gerenciamento é um dos fatores mais importantes no projeto e implementação dos sistemas operacionais. Enquanto nos sistemas monoprogramáveis, a gerência de memória não é muito complexa, nos sistemas multiprogramáveis, essa gerência se torna crítica, devido à necessidade de se maximizar o número de usuários e aplicações utilizando eficientemente o espaço da memória principal.

Geralmente, os programas são armazenados em memórias secundárias, de uso permanente e não voláteis, como discos ou fitas. Como o processador somente executa o que está na memória principal, o sistema operacional deve sempre transferir programas da memória secundária para a principal antes de executá-los. Como o tempo de acesso às memórias secundárias é muito superior ao tempo de acesso à memória principal, o sistema operacional busca reduzir o número de operações de E/S (acessos à memória secundária) a fim de não comprometer o desempenho do sistema.

A gerência de memória deve, então, tentar manter na memória principal o maior número de processos residentes, permitindo maximizar o compartilhamento do processador e demais recursos computacionais. Mesmo que não haja espaço livre, o sistema deve permitir que novos processos sejam aceitos e executados. Outra preocupação na gerência de memória é permitir a execução de programas maiores do que a memória física disponível.

Em um ambiente de multiprogramação, o sistema operacional deve proteger as áreas de memória ocupadas por cada processo, além da área onde reside o próprio sistema. Caso um programa tente realizar algum acesso indevido à memória, o sistema deve, de alguma forma, impedir o acesso. A seguir, você verá as principais técnicas de alocação de memória utilizadas pelo gerenciador de memória:

- *Alocação contígua simples*: esse tipo de alocação foi implementado nos primeiros sistemas operacionais, embora nos dias atuais ainda esteja presente em alguns sistemas monoprogramáveis. Nesse modelo, a memória principal é dividida em duas partes, uma para o sistema operacional e outra para o programa do usuário. Dessa forma, o programador desenvolve suas aplicações preocupado apenas em não ultrapassar o espaço de memória disponível.

- *Segmentação*: na alocação contígua simples todos os programas estão limitados ao tamanho da memória principal disponível para o usuário. Uma solução encontrada para o problema é dividir o programa em módulos, de modo que seja possível a execução independente de cada módulo, utilizando a mesma área de memória. A essa técnica dá-se o nome de *segmentação* ou *overlay*.

- *Alocação particionada estática*: os sistemas operacionais evoluíram no sentido de proporcionar melhor aproveitamento dos recursos disponíveis. Nos sistemas monoprogramáveis, o processador permanece grande parte do tempo ocioso e a memória principal é subutilizada. Os sistemas multiprogramáveis são mais eficientes no uso do processador, necessitando que vários processos estejam na memória principal ao mesmo tempo e que novas formas de gerência de memória sejam implementadas. Nos primeiros sistemas multiprogramáveis, a memória era dividida em blocos de tamanho fixo, chamados *partições*. O tamanho dessas partições, estabelecido em tempo de inicialização do sistema, era definido em função do tamanho dos programas que seriam executados no ambiente. Sempre que fosse necessária a alteração do tamanho de uma partição, o sistema deveria ser inicializado novamente com uma nova configuração.

- *Alocação particionada dinâmica*: a alocação particionada estática deixou clara a necessidade de uma nova forma de gerência de memória principal, onde o problema da fragmentação interna fosse reduzido e, consequentemente, o grau de compartilhamento da memória aumentado. Na alocação particionada dinâmica, foi, então, eliminado o conceito de partições de tamanho fixo. Nesse esquema, cada programa, ao ser carregado, utilizaria o espaço necessário à sua execução, tornando esse espaço a sua partição. Assim, como os programas utilizam apenas o espaço de que necessitam, no esquema de alocação particionada dinâmica o problema da fragmentação interna deixa de existir.

Várias das técnicas anteriores podem utilizar um mecanismo muito importante, chamado *swapping*. Trata-se de uma técnica aplicada à gerência de memória que visa dar maior taxa de utilização à memória principal, melhorando seu compartilhamento. Visa também resolver o problema da falta de memória principal em um sistema.

Toda vez que um programa precisa ser alocado para execução e não há espaço na memória principal, o sistema operacional escolhe um processo entre os alocados que não possuem previsão de utilizar a CPU nos próximos instantes (quase sempre entre aqueles que estão em interrupção de E/S ou no final da fila de pronto), "descarregando" esse processo da memória para uma área especial em disco, chamada *arquivo de swap*, onde o processo fica armazenado temporariamente. Durante o tempo em que o processo fica em *swap*, o outro que necessitava de memória entra em execução, ocupando o espaço deixado pelo que saiu.

Pouco antes de chegar a vez do processo armazenado em *swap* utilizar a CPU, o sistema escolhe outro processo para descarregar para *swap* e devolve o anterior da área de *swap* para a memória principal, a fim de que esse seja executado novamente. Trabalha-se dessa forma até que os processos terminem. O problema dessa técnica é a possibilidade de se provocar um número excessivo de acessos à memória secundária (disco), levando o sistema a uma queda de desempenho.

Acabamos de observar as diversas técnicas de gerenciamento de memória que, ao longo do tempo, evoluíram no sentido de maximizar o número de processos residentes na memória principal e reduzir o problema da fragmentação, porém os esquemas aqui apresentados se mostraram, na maior parte dos casos, ineficientes. Além disso, o tamanho dos programas e de suas estruturas de dados estava limitado ao tamanho da memória disponível. Como vimos, a utilização da técnica de *overlay* para contornar esse problema é de difícil implementação na prática e nem sempre se apresenta como uma solução garantida e eficiente.

Não se esqueça que *memória virtual* é uma técnica sofisticada e poderosa de gerência de memória, onde a memória principal e a memória secundária são combinadas, dando ao usuário a impressão de que existe muito mais memória do que a capacidade real de memória principal. O conceito de memória virtual se baseia em não vincular o endereçamento feito pelo programa aos endereços físicos da memória principal. Dessa forma, o programa e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços vinculados à memória secundária, que, por sua vez, funciona como uma extensão da memória principal. Outra vantagem dessa técnica é permitir um número maior de processos, compartilhando a memória principal, já que apenas partes de cada processo estarão residentes. Isso leva a uma utilização mais eficiente do processador, além de minimizar (ou quase eliminar) o problema da fragmentação.

5. Gerenciamento de arquivos

Um sistema operacional, normalmente, organiza seus arquivos por meio de *diretórios*. Um diretório contém entradas associadas aos arquivos, com as informações de localização, nome, organização e outros atributos. Podem ser organizados da seguinte forma:

- *Nível único*: é a implementação mais simples de uma estrutura de diretórios, onde existe um único diretório contendo todos os arquivos do disco. É muito limitado, não permitindo a criação de arquivos com o mesmo nome.
- *Diretório pessoal*: evolução do modelo anterior, permite a cada usuário ter seu "diretório" particular, sem a preocupação de conhecer os outros arquivos do disco. Nesse modelo, há um diretório "*master*" que indexa todos os diretórios particulares dos usuários, provendo o acesso a cada um desses.

- *Múltiplos níveis (ÁRVORE)*: é o modelo, atualmente, utilizado em quase todos os sistemas operacionais. Nessa modalidade, cada usuário pode criar vários níveis de diretórios (ou subdiretórios), sendo que cada diretório, por sua vez, pode conter arquivos e subdiretórios. O número de níveis possíveis depende do sistema operacional.

Arquivos e diretórios são organizados, em um nível mais elevado de abstração, como um sistema de arquivos (*file system*). São exemplos importantes de sistemas de arquivos:

- *FAT*: sistema criado no *MS-DOS* e depois utilizado no *Windows*. Usa listas encadeadas e possui um limite de área utilizável, em partições, de 2GB. Caracteriza-se por um baixo desempenho no acesso e armazenamento.
- *FAT32*: igual ao *FAT* no que diz respeito à organização e desempenho, contudo, pode trabalhar com partições de até 2TB e nomes para arquivos com tamanhos variados.
- *NTFS*: *NT File System*, original da plataforma *Windows NT/2000/XP*. Opera com uma estrutura em árvore binária, oferecendo alto grau de segurança e desempenho.
- *UNIX:* usa diretório hierárquico, com uma raiz e outros diretórios subordinados. Nesse sistema operacional, todos os arquivos são considerados apenas como uma "sequência" de *bytes*, sem significado para o sistema. A aplicação é responsável pelo controle dos métodos de acesso aos arquivos. O *UNIX* também utiliza alguns diretórios padronizados, de exclusividade do sistema.

Um problema muito importante para o gerenciador de arquivos é como administrar o espaço livre dentro de um disco. São três as formas de se implementar estruturas de espaços livres. Uma delas é por meio de uma tabela denominada *mapa de bits*, onde cada entrada dessa tabela é associada a um bloco do disco, representado por um *bit*, que, estando com valor 0, indica que o espaço está livre. Consequentemente, o valor 1 representa um espaço ocupado. Ocupa muita memória, pois para cada bloco do disco há uma entrada na tabela.

A segunda forma se dá na utilização de uma *lista encadeada pelos blocos livres* do disco. Desse modo, cada bloco possui uma área reservada para armazenar o endereço do próximo bloco livre. Esse caso apresenta problemas de lentidão no acesso, devido às constantes buscas sequenciais na lista.

Por sua vez, a terceira forma é a *tabela de blocos livres*, que leva em consideração o fato de que blocos contíguos de dados, geralmente, são alocados/liberados simultaneamente. Dessa forma, pode-se enxergar o disco como um conjunto de segmentos de blocos livres. Assim, é possível manter uma tabela com o endereço do primeiro bloco de cada segmento e o número de blocos contíguos que se seguem. Finalmente, um detalhe importante que deve ser contemplado por um gerenciador de arquivos: a segurança de acesso aos arquivos e diretórios. Considerando que os meios de armazenamento são compartilhados por vários usuários, é fundamental que mecanismos de proteção sejam implementados para garantir a integridade e proteção individual dos arquivos e diretórios. Dos mecanismos possíveis, seguem algumas possibilidades:

- *Senha de acesso*: mecanismo de simples implementação, contudo, apresenta duas desvantagens: não possibilita determinar quais os tipos de operação podem ser efetuadas no arquivo e, se esse for compartilhado, todos os usuários que o utilizam devem conhecer a senha de acesso.

- *Grupo de usuários*: muito utilizado em diversos sistemas operacionais. Consiste na associação de cada usuário a um grupo. Esses grupos são organizados logicamente com o objetivo de compartilhar arquivos e diretórios no disco. Esse mecanismo implementa três níveis de proteção: *OWNER* (dono), *GROUP* (grupo) e *ALL* (todos). Na criação do arquivo, o usuário especifica se esse poderá ser acessado somente pelo seu criador, pelo grupo ou por todos os demais usuários, além de definir quais tipos de acesso podem ser realizados (leitura, escrita, execução e eliminação).
- *Lista de controle de acesso*: trata-se de uma lista associada ao arquivo, onde são especificados quais os usuários e os tipos de acesso permitidos. O tamanho dessa estrutura pode ser bastante extenso se considerarmos que um arquivo chega a ser compartilhado por vários usuários. Além desse problema, há o inconveniente de se fazer acesso sequencial à lista a toda momento em que um acesso é solicitado. Em determinados sistemas de arquivos, é possível utilizar uma combinação de proteção por grupos de usuários ou por listas de acesso, oferecendo, assim, maior flexibilidade ao mecanismo de proteção desses arquivos e os respectivos diretórios.

6. Gerenciamento de dispositivos

Para realizar funções úteis, processos precisam acessar periféricos conectados ao computador, que são controlados pelo núcleo por meio do *driver do dispositivo*. Um exemplo: para mostrar ao usuário algo utilizando a tela, um aplicativo teria que fazer uma requisição ao núcleo que, por sua vez, encaminharia a requisição para o seu *driver* de tela, que é responsável por, realmente, tracejar os caracteres/*pixeis*.

O núcleo de um sistema operacional deve manter uma lista de dispositivos disponíveis. Essa lista pode ser conhecida de antemão (um sistema embarcado, onde o núcleo será reescrito se o *hardware* disponível mudar é um exemplo), configurado pelo usuário (ação típica em computadores pessoais antigos e em sistemas projetados para uso pessoal) ou detectado pelo sistema durante a execução do próprio dispositivo (habitualmente chamado *plug-and-play*).

Em um sistema do tipo *plug-and-play*, o dispositivo, inicialmente, realiza uma sondagem nos diferentes barramentos de *hardware* para detectar os dispositivos já instalados para, só depois, buscar os *drivers* apropriados para sua instalação.

Como o gerenciamento de dispositivos é um tópico muito especifico do sistema operacional, esses *drivers* são manipulados de modos diferentes por cada tipo de projeto de núcleo, contudo, em todos os casos, o núcleo tem que fornecer a entrada/saída para permitir que os *drivers* acessem fisicamente seus dispositivos por meio de alguma porta ou localização da memória.

Decisões muito importantes precisam ser tomadas ao projetar o sistema de gerenciamento de dispositivos, já que alguns projetos de acesso podem envolver trocas de contexto, tornando a operação complexa para o processador, além de causar um gasto excessivo de recursos.

Material Complementar

Conheça mais sobre o sistema operacional *Windows*:

Disponível em: http://www.microsoft.com. Acesso em: 30 mar. 2010.

Conheça mais sobre o sistema operacional *Linux*:

Disponível em: http://www.linux.org. Acesso em: 30 mar. 2010.

Conheça mais sobre o sistema operacional *MacOS*:

Disponível em: http://www.apple.com. Acesso em: 30 mar. 2010.

Referências

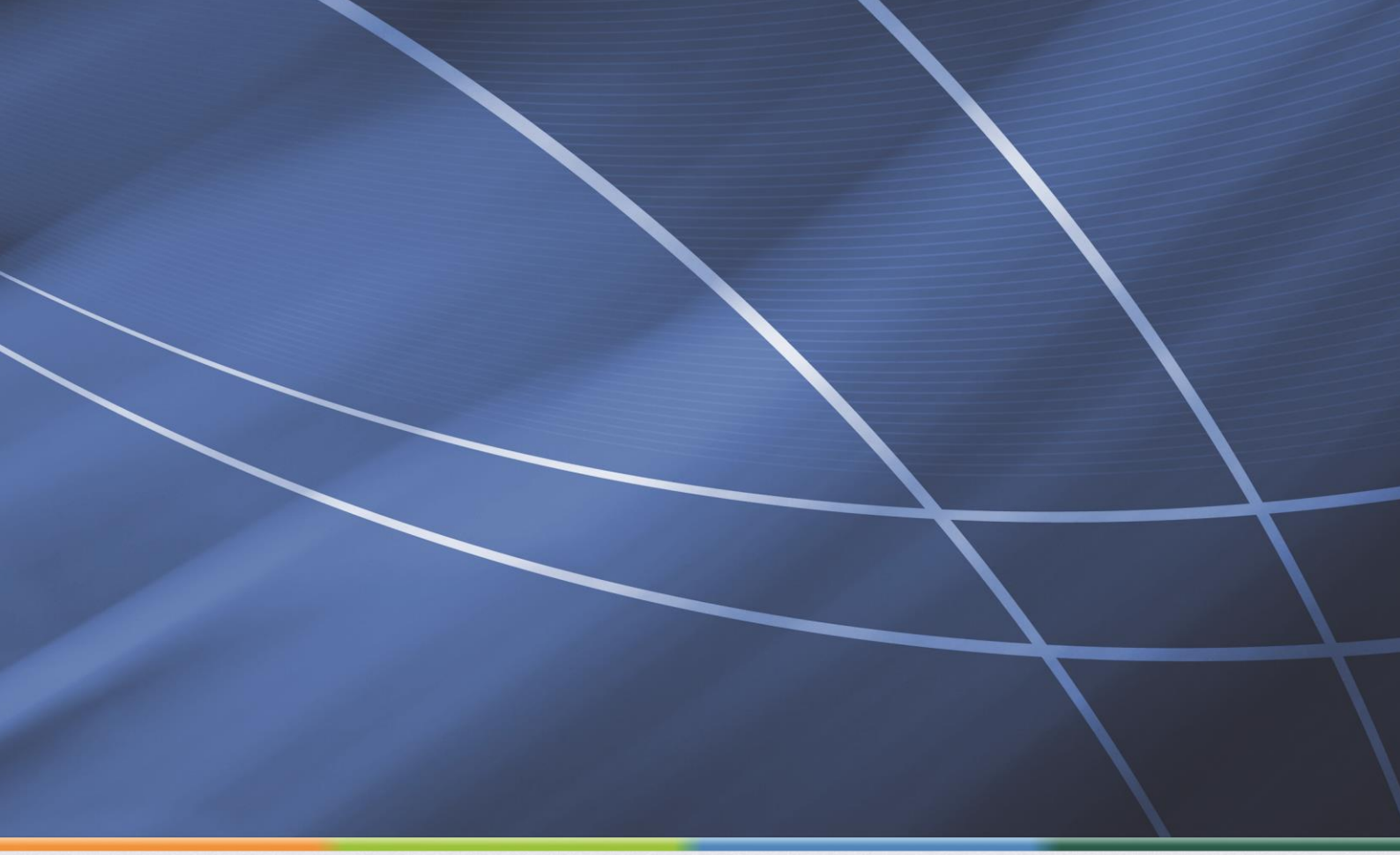
MAIA, L. P.; MACHADO, F. B. **Arquitetura de sistemas operacionais**. 4. ed. Rio de Janeiro: LTC, 2007.

MMOKARZEL, F.; SOMA, N. **Introdução à ciência da computação**. São Paulo: Campus, 2008.

Anotações

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade Cruzeiro do Sul

Módulo Centro Universitário

UDF Centro Universitário

UNICID Universidade Cidade de S. Paulo