INTRODUÇÃO AOS SISTEMAS OPERACIONAIS agosto/2013

SEMANA 06

Estrutura dos Sistemas Operacionais: Componentes do sistema, serviços, chamadas de sistema.

1 - Introdução

A estrutura do sistema operacional é formada por um conjunto de rotinas e procedimentos com o objetivo de oferecer serviços às aplicações e usuários. Essa estrutura também atende a processos de sistema, que são independentes de usuário. Esse conjunto de rotinas com funções específicas estão no kernel ou núcleo do sistema operacional.

O kernel não é visto pelos usuários: aquilo que o usuário vê são os utilitários do sistema operacional. Por exemplo, o Windows Explorer é um utilitário, e como tal não passa de uma aplicação específica do Windows. São os utilitários que permitem aos usuários e administradores de sistema interagir com o sistema operacional. Um utilitário é uma aplicação normalmente instalada junto com o sistema operacional.

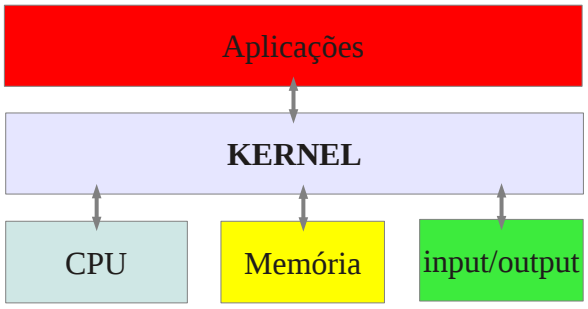

Fig 1 – Aplicações, kernel e hardware

O principal propósito do kernel consiste no gerenciamento dos recursos do computador, para permitir que as aplicações rodem e usem estes recursos. Tipicamente, estes recursos consistem de Unidade Central de Processamento [CPU], memória, dispositivos de entrada/saída [Input/Output, I/O], processos e sistema de arquivos.

Além disso, o kernel é a última camada de software [a de nível mais baixo] que pode oferecer uma camada de abstração para os recursos de hardware [especialmente processadores e dispositivos de entrada/saída] que os softwares aplicativos devem controlar para realizar as suas funções. O kernel disponibiliza estas facilidades através de mecanismos de comunicação entre processos e chamadas de sistema.

Então, kernel é simplesmente o nome dado ao nível mais inferior de abstração que é implementado em software.

A CPU, Unidade Central de Processamento, é a parte central do computador, responsável por *rodar* ou *executar as aplicações*. O kernel tem a responsabilidade de decidir em qualquer momento qual dos processos em execução deve ser escalonado [alocado] para o processador.

A memória é usada para armazenamento de instruções e dados do programa. Tipicamente, tanto instruções quanto dados precisam estar presentes na memória para tornar possível a execução do programa. Muitas vezes, alguns programas buscarão acesso à memória simultaneamente, em alguns casos até exigirão mais memória do que o computador pode disponibilizar. É o kernel que decide sobre a memória que cada processo pode utilizar, e determina o que fazer quando não há memória suficiente disponível.

Os dispositivos de entrada e saída [I/O] são mouse, monitor, teclado, impressora, disco, placa de rede, pendrive, etc. O kernel recebe pedidos das aplicações para realizar entrada/saída para os dispositivo [ou mesmo subseção de dispositivos, por exemplo arquivos em disco ou janelas em uma tela] e fornece métodos convenientes para o uso desses dispositivos. Para as aplicações que fazem esses pedidos ao kernel, não é necessário conhecer os detalhes da implementação de tais dispositivos.

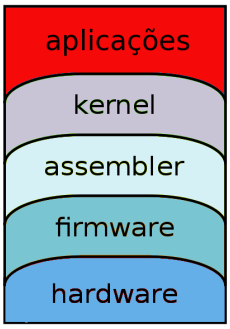

Fig 2 – Camadas de abstração

O sistema operacional não é aplicação nem utilitário, mas um software que roda no modo kernel ou modo supervisor, e tem por objetivo proteger o hardware da ação direta do usuário.

Existem algumas instruções que não podem ser disponibilizadas diretamente às aplicações, pois se fossem utilizadas indevidamente ocasionariam sérios problemas à integridade do sistema. Essas instruções, que podem comprometer o sistema, são conhecidas como instruções privilegiadas. As instruções não-privilegiadas são as que não oferecem risco à integridade do sistema.

Existem dois modos de privilégios para execução na CPU: o modo supervisor [kernel mode] e o modo de usuário [user mode]. Essas são camadas de proteção [protection rings] que, para serem implementadas, precisam também serem suportadas pelo processador.

Normalmente a inicialização do sistema, no boot, executa o kernel no modo supervisor [kernel modo]. No modo supervisor, o kernel tem total controle sobre a máquina e é assim que gerencia os processos.

O funcionamento normal é no modo de usuário, onde o kernel não executa diretamente mas apenas em resposta a eventos externos, como chamadas de sistema usadas pelos aplicativos para requisitar serviços ao kernel ou interrupções usadas pelo hardware para notificar o kernel sobre eventos. Desse modo, os processos executam em modo usuário e por isso não detêm o controle total da máquina.

Os demais componentes de software do sistema, como os compiladores e editores rodam no modo usuário. Do mesmo modo, aplicações como Firefox, Excel e Outlook também rodam no modo usuário.

Portanto, apenas o kernel tem o controle total da máquina. Já os processos que rodam nesse sistema operacional executam em modo usuário e por isso não detêm o controle total da máquina.

Quando o processador trabalha no modo usuário, a aplicação só pode executar instruções não-privilegiadas, e por isso tem acesso a um número reduzido de instruções. No modo kernel, essa aplicação pode ter acesso ao conjunto total de instruções do processador.

Para o sistema verificar se a instrução pode ou não ser executada por determinada aplicação, existe um conjunto de bits localizados no registrador de status do processador [*PSW*, *Processor Status Word*], que indica o modo de acesso corrente dessa aplicação.

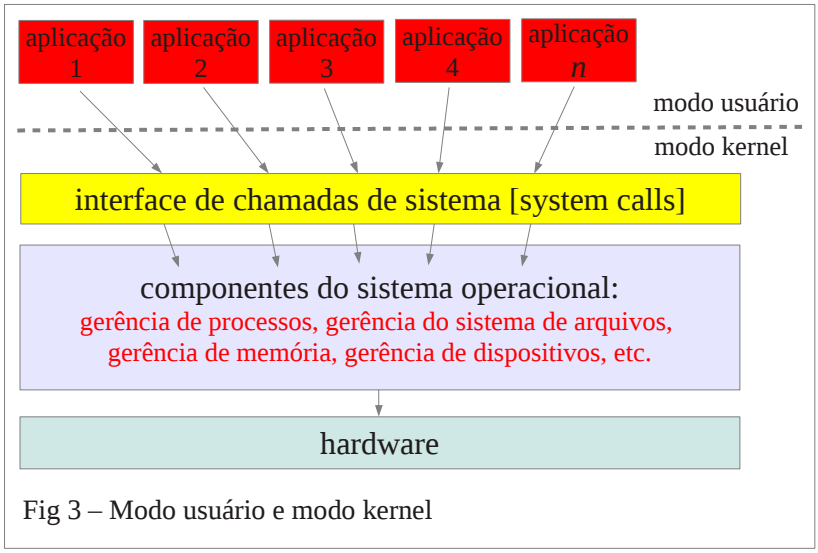

Fig 3 – Modo usuário e modo kernel

Se a aplicação necessitar executar uma instrução privilegiada, a solicitação deve ser realizada através de uma chamada de sistema [system call] que altera o modo de acesso do processador do modo usuário para o modo kernel. E se essa aplicação tentasse executar determinada instrução privilegiada no modo usuário, o processador sinalizaria um erro, seria gerada uma exceção e interrompida a execução dessa aplicação.

Além do modo usuário e modo kernel, atualmente o kernel também pode trabalhar num terceiro modo chamado de hipervisor [hypervisor mode]. O modo hipervisor tem total controle da CPU, e serve para o monitor de máquinas virtuais gerenciar os sistemas operacionais abrigados em suas máquinas virtuais. Enquanto a interface entre modo usuário e modo supervisor ainda é a mesma, essa é uma nova interface entre o modo supervisor e o hardware. Desse modo, todas as instruções privilegiadas feitas pelo kernel são capturadas pelo monitor de máquinas virtuais [administrative operating system and/or management console] que pode emular o comportamento desejado. Este método é chamado captura e emulação [trap-and-emulate].

Diferente do hipervisor tipo 2, o hipervisor tipo 1 roda direto no hardware [bare metal], sem um sistema operacional. Na prática, nesse caso o próprio hipervisor é um kernel modificado. Como exemplos, temos Xen, VMware ESX ou ESXi Server e Microsoft Hyper-V technology.

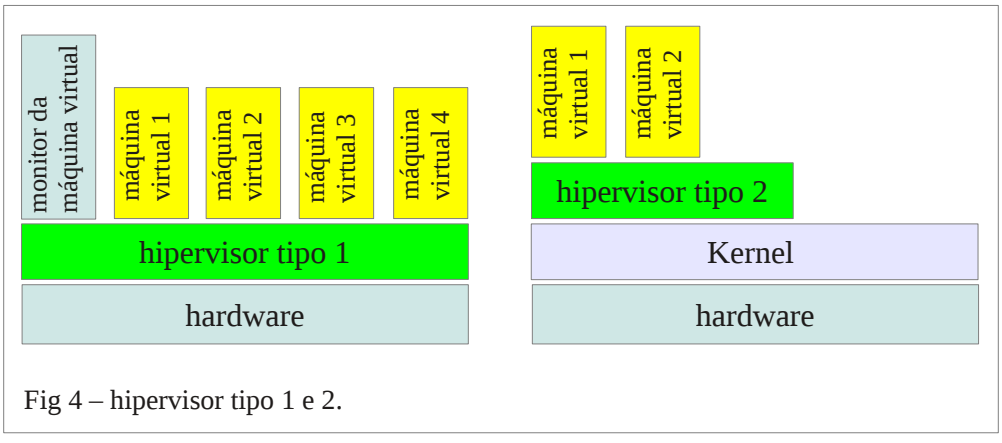

Fig 4 – hipervisor tipo 1 e 2.

Pelo que foi visto acima, é também o kernel que arbitra as requisições das aplicações que concorrem pelos recursos de hardware.

A estrutura do sistema operacional é a maneira como o código do sistema está organizado no interrelacionamento dos seus diversos componentes. Quanto à estrutura dos sistemas operacionais,

estas se dividem em **componentes do sistema**, **serviços** e **chamadas de sistema**.

No entanto, essa estrutura pode variar conforme a concepção do projeto do sistema operacional.

2 - Componentes do sistema operacional

Basicamente, o núcleo do sistema operacional executa as suas funções a partir de gerentes. Os principais componentes ou funções do kernel são:

- 2.1 - tratamento das interrupções e exceções;
- 2.2 - gerenciamento de processos e threads;
- 2.3 - gerenciamento de memória;
- 2.4 - gerenciamento do sistema de arquivos;
- 2.5 - gerenciamento dos dispositivos de entrada/saída;
- 2.6 - suporte a redes locais e distribuídas.

2.1 - Tratamento das interrupções e exceções

As interrupções alteram o fluxo normal do programa para atender a eventos externos ou reportar erros ou condições de exceção. Essas interrupções podem ser de software ou hardware.

Como exemplo de exceções, se uma aplicação tentar efetuar uma divisão por zero será gerada uma exceção que vai interromper a execução dessa aplicação.

Quando o usuário pressiona uma tecla, uma interrupção é sinalizada à CPU, e se tudo estiver correto a CPU irá parar o que estiver rodando e chamar uma função que vai ler a porta do teclado para descobrir o que está sendo enviado. Essa função também precisa posteriormente retornar o que estivesse rodando antes que a interrupção ocorresse. Nesse caso, o código que estava rodando e foi parado pela interrupção, ao retornar o processamento não tem como saber que houve essa interrupção.

Além do teclado, muitos outros dispositivos também usam interrupções. HD, disquete, pendrive, placa de som, DVD, placa de rede e modem usam interrupções para dizer ao sistema operacional que completaram alguma coisa ou informar que existem dados a serem enviados.

2.2 - Gerenciamento de processos e threads

Numa definição de processo, temos que este é um programa em execução. Para executar necessita de recursos como CPU, memória, acesso a sistema de arquivo e dispositivos de entrada/saída [I/O].

A tarefa de gerenciamento de processos envolve:

- criação e exclusão de processos;

- suspensão e retorno à execução;
- sincronismo e comunicação entre processos.

2.3 - Gerenciamento da memória

A função do gerenciador de memória é alocar espaços de memória a cada programa a ser executado, e liberar esses espaços uma vez terminada a execução. A memória é um dos principais recursos a ser controlado pelo sistema operacional, e em sistemas que permitem multitarefa é necessário fornecer memória para os novos processos, à medida que eles são criados.

O gerenciador de memória [*memory manager*] controla a alocação e liberação de memória para os processos, além de gerenciar a troca de informações entre a memória principal e o sistema de arquivos [swapping].

Um programa carregado na memória tipicamente apresenta as seguintes divisões:

- área de dados;
- área de código;
- área de pilha (stack e heap[1]).

A maioria das linguagens de programação permite a construção de estruturas de dados dinâmicas através de alocação dinâmica de memória como, por exemplo, o comando *malloc* em C e a primitiva *new* em Pascal, e estas requisições de alto nível devem ser atendidas pelo gerenciador de memória do sistema.

Outra questão interessante reside na possibilidade de reutilização de código já carregado em memória, também chamado de código reentrante ou público (*public code*). Esta porção da memória pode ser compartilhada entre vários processos simultâneos, geralmente contendo arquivos executáveis. Entretanto, deve-se considerar o custo de controle inerente a este mecanismo.

2.3.1 - Monoprogramação pura

A maneira mais simples de gerenciar a memória consiste em permitir que apenas um processo por vez esteja na memória, e permitir que esse processo utilize parte ou toda a memória disponível. Nesse caso, a alocação pode ser contígua ou não.

O caso da alocação não contígua é mais complexo, pois utiliza espaços não ocupados e fragmentados de memória.

Um programa maior do que a memória primária disponível pode ser executado através da técnica de sobreposição [*overlay*]: o programador divide o programa em módulos independentes que são trazidos da memória secundária [mídia de armazenamento, HD] para a memória principal a partir de sua necessidade.

1 Normalmente tanto *stack* quanto *heap* são traduzidos como pilha, embora tratem de coisas distintas. O mais correto seria tratar *stack* como pilha e *heap* como "monte". Quase ninguém usa a tradução "monte".

2.3.2 - Multiprogramação com partições fixas

No caso da multiprogramação, vários processos estão competindo simultaneamente pelo uso de recursos de memória primária. A multiprogramação com partições fixas consiste em dividir a memória em um número qualquer de partições, de tamanhos iguais ou diferentes.

Nesse esquema, pode-se então ter uma fila de entrada para cada partição, e os processos permanecerem esperando para serem servidos pelo gerenciador de memória. Neste caso, pode haver desperdício de memória, como é o caso de uma partição grande sempre vazia.

Já outra possibilidade reside em manter uma fila única de entrada que contemple todas as partições: assim que uma partição se tornar livre o gerenciador de memória escolhe e carrega a tarefa mais próxima do início da fila, desde que caiba nessa partição.

Como os processos interativos são geralmente menores e devem ter tempos de resposta mais rápidos devido a sua maior prioridade, pode-se ter sempre disponível uma partição pequena para atender a estas tarefas e assim evitar desperdício de memória através da utilização de partições maiores.

2.3.3 - Multiprogramação com partições variáveis

Com o intuito de maximizar a utilização da memória foi criada a multiprogramação com partições variáveis. A diferença principal em relação ao esquema de partições fixas é que o número, localização e tamanho das partições varia dinamicamente dependendo da execução dos processos.

Porém, a otimização da memória implica em maior complexidade dos algoritmos de alocação e liberação de memória, pois a todo instante processos são criados e finalizados gerando o problema conhecido como fragmentação da memória.

A principal vantagem na utilização de partições variáveis é que somente a memória necessária a um processo é de fato alocada. Porém, alguns inconvenientes podem surgir, pois em geral os processos possuem área de dados dinâmicas além de *stack* e *heap* que podem aumentar durante sua execução, havendo necessidade constante de realocação da memória, com isso causando uma sobrecarga (*overhead*).

Tanto *stack* quanto *heap* são áreas de memória na RAM, sendo que *stack* é mais rápido de alocar que *heap*. *Stack* é usualmente pré-alocado [ao iniciar a aplicação] e por definição precisa estar num espaço contíguo de memória. *Heap* pode ser alocado dinamicamente, à medida que a aplicação roda.

No caso do processo possuir dados dinâmicos, pode-se contornar o problema de *overhead* alocando previamente uma pequena porção extra de memória.

2.3.4 - Multiprogramação com swapping

Nas técnicas de multiprogramação anteriores os processos permaneciam na memória primária até o final de sua execução. A estratégia de *swapping* faz com que um processo possa ser

retirado da memória primária e armazenado no disco, permitindo que outros programas utilizem essa memória. Em sistemas multitarefa a quantidade de memória primária disponível pode não ser suficiente para comportar todos os processos de uma só vez, o que justifica o uso dessa técnica.

2.3.5 - Realocação de endereços e proteção de memória

A partir do uso da multiprogramação, dois problemas surgem: a *realocação* e a *proteção*.

A realocação envolve a conversão dos endereços relativos de memória utilizados durante a execução de um programa, e proteção consiste em evitar que os processos de um usuário leiam e escrevam em posições de memória de outros usuários.

2.3.6 - Memória virtual

Neste método a memória disponível não é utilizada diretamente através de endereços reais, mas sim mapeada em endereços virtuais.

Como cada processo possui um endereçamento virtual distinto e deve existir uma função de mapeamento que traduza os endereços virtuais [utilizados pelos processos] em endereços reais [disponíveis no hardware], o sistema operacional controla as partes necessárias à execução do processo que devem estar presentes na memória primária.

O endereçamento virtual permite que o sistema utilize meios de armazenamento de dados, como disco, para armazenar o que de outro modo teria que permanecer na memória RAM. Isso possibilita que aplicações usem mais memória do que está disponível fisicamente. Quando a aplicação A precisar de dados que não estão na RAM, o kernel pode transferir o conteúdo de um bloco de memória inativo [pertencente a aplicação B] para o disco, e então aproveitar esse espaço de RAM disponibilizada para gravar os dados requisitados pela aplicação A. Agora, a aplicação A pode continuar a sua execução do ponto em que foi suspensa. Este esquema é geralmente conhecido como paginação por demanda.

O uso de memória virtual permite, por exemplo, que uma aplicação que demande 2,5 GB possa executar em uma máquina com 2 GB de memória primária, desde que tenha uma área de swap maior que 500 MB [nesse exemplo].

Endereçamento virtual também permite a criação de partições virtuais de memória em duas áreas separadas, uma sendo reservada para o kernel [espaço de kernel] e o outro para os aplicativos [espaço de usuário]. Os aplicativos não tem permissão do processador para acessar a memória do kernel, portanto prevenindo que um aplicativo possa danificar o kernel em execução.

As duas principais técnicas de memória virtual são *paginação* e *segmentação*.

2.3.7 - Paginação

Nesta técnica de gerenciamento de memória, o espaço de endereçamento virtual é dividido em páginas [*pages*] de tamanho fixo [geralmente entre 512 bytes até 8KB], e o espaço real de

endereçamento [memória física] é dividido em quadros de página [*page frames*] de mesmo tamanho que as páginas virtuais.

As transferências de dados de processos entre o sistema de arquivo [disco, pendrive, cartão de memória] e a memória é chamado de *swap* e feito em unidades de página. A página virtual possui um *bit* de presença que indica se ela já está carregada na memória. Caso determinada página não esteja na memória, esta é carregada do sistema de arquivo para a memória em um quadro livre, e se uma aplicação referenciar uma página que não esteja na memória principal ocorre uma falta de página [*page fault*]. Se não houver páginas livres na memória principal o sistema operacional deve escolher qual página será enviada de volta para o sistema de arquivo.

Para determinar qual página será enviada para o sistema de arquivo, diversas estratégias podem ser utilizadas: NRU [*Not Recently Used*], FIFO [*First In, First Out*], LRU [*Least Recently Used*], entre outras.

As páginas virtuais são traduzidas em páginas reais através de uso de uma tabela de página [*page table*], em que existe um relacionamento entre endereços lógicos [virtuais] e físicos [reais].

2.3.8 - Segmentação

No esquema anterior de paginação, as páginas possuem tamanhos fixos. Para resolver este obstáculo criou-se o mecanismo de segmentação que consiste em dividir a memória em segmentos que podem aumentar e diminuir de tamanho durante a execução dos processos.

Para isso, os compiladores dividem o código do programa em segmentos distintos [pilha, dados, código, funções], que são armazenados em posições de memória diferentes.

O esquema de mapeamento é similar ao da paginação, e normalmente os sistemas operacionais utilizam segmentação com paginação.

2.4 - Gerenciamento do sistema de arquivos

Usualmente, a parte mais visível de um sistema operacional é o seu sistema de arquivo. Através de chamadas de sistema, programas aplicativos utilizam o sistema de arquivo para criar, ler, gravar e remover arquivos.

A conveniência e facilidade de uso de um sistema operacional do tipo *desktop* ou estação de trabalho [*workstation*] é fortemente determinada pela interface, estrutura e confiabilidade de seu sistema de arquivos. Do ponto de vista do usuário, o aspecto mais importante do sistema de arquivo é como ele se apresenta, isto é, o que constitui um arquivo, como os arquivos são identificados e protegidos, que operações são permitidas sobre os arquivos, e assim por diante.

Já no sistema operacional do tipo servidor, o aspecto mais importante se relaciona à disponibilidade, rapidez e segurança dos arquivos.

Para o projetista do sistema, por outro lado, a preocupação é como organizar e gerenciar

arquivos de forma a garantir boa ocupação do espaço de armazenamento de dados e bom desempenho nas operações sobre arquivos.

2.4.1 - Interface do sistema de arquivo

A maior parte dos sistemas operacionais traz a seguinte proposta para armazenamento de informação: permitir aos usuários definir objetos chamados arquivos, que podem armazenar programas, dados, ou qualquer outra informação.

Estes arquivos não são parte endereçável de nenhum processo, e o sistema operacional suporta chamadas de sistema para criar, excluir, ler, atualizar e proteger os arquivos.

A maior parte dos sistemas operacionais suporta vários tipos de arquivos. Por exemplo, arquivos regulares [ou ordinários], diretórios e arquivos especiais. Arquivos regulares contém dados e programas do usuário. Diretórios são meras estruturas organizacionais e arquivos especiais são usados para especificar periféricos tais como terminais, impressoras e unidades de fita.

Em muitos sistemas, arquivos regulares são subdivididos em diferentes tipos em função de sua utilização. Os tipos são identificados pelos nomes com que os arquivos regulares terminam. Por exemplo, "arquivo.c" denota um programa fonte em C e "arquivo.obj" para um arquivo objeto.

Em alguns sistemas, como é o caso dos membros da família Unix, as extensões ou não são usadas ou são simples convenções e que representam apenas uma facilidade para o usuário identificar o tipo de conteúdo no arquivo. Em outros, o sistema operacional tem regras rígidas em relação aos nomes, por exemplo exigir que o executável tenha a extensão ".EXE". Esse último é o caso do Windows.

Diretórios permitem organizar os arquivos de um sistema. Um diretório contém tipicamente um registro por arquivo. Sistemas primitivos admitiam um único diretório compartilhado por todos os usuários ou um único diretório por usuário, já os sistemas operacionais modernos permitem um número arbitrário de diretórios por usuário, em geral numa estrutura hierárquica.

2.4.2 - Gerência de espaço em disco

Arquivos são normalmente armazenados em disco[2], sendo portanto a gerência do espaço em disco de maior importância no sistema operacional. Duas estratégias são possíveis para armazenamento em um arquivo com *n bytes*: *n bytes* consecutivos do disco são alocados ou então o arquivo é dividido em um número de blocos não necessariamente contíguos.

Quando blocos de tamanho fixo são adotados, é necessário definir qual o tamanho do bloco. Uma unidade de alocação grande, tal como uma trilha, implica em que muitos arquivos, até mesmo arquivos de 1 byte, deverão ocupar a trilha inteira. Por outro lado, usar uma unidade de alocação pequena significa que cada arquivo terá muitos blocos, o que pode prejudicar o desempenho de acesso.

2 Mais recentemente, o disco está começando a perder espaço para o *flash memory*, que é um tipo de memória de estado sólido usada em pendrives, cartões de memória e até HDs.

É compromisso usual escolher um bloco de tamanho entre 512 e 8K bytes. Se um bloco de tamanho 1K for escolhido em um disco com setor de 512 bytes, então o sistema de arquivos sempre irá ler ou escrever em dois setores contíguos, e tratá-los como uma unidade indivisível.

A escolha da estratégia de alocação de um arquivo no disco traz impactos no desempenho e flexibilidade de acesso aos dados armazenados.

2.5 - Gerência dos dispositivos de entrada/saída

Para que possam realizar as suas funções, os processos precisam ter acesso aos periféricos conectados ao computador. Esse acesso vem através do kernel. No Windows, por exemplo, usa-se o driver de dispositivo [software] para acesso ao periférico [hardware].

2.5.1 - Unidade de entrada e saída [I/O]

Os dispositivos de I/O podem ser divididos em três categorias: os que recebem ou transmitem informações diretamente para o ser humano [teclado, vídeo, impressora, etc.], os que recebem ou transmitem informações inteligíveis apenas pela máquina [discos magnéticos, discos óticos, sensores] e os que recebem e transmitem de/para dispositivos remotos [modem, regeneradores de sinais].

Quanto à forma de se estabelecer a ligação, esta pode ser serial [a informação flui *bit* a *bit*] ou paralela [vários bits são transmitidos simultaneamente]. A ligação periférico/CPU, além dos dados em si, também compreende alguns sinais de controle, cujas características também variam muito de um para outro periférico.

2.5.2 - Sistema de entrada/saída [I/O]

Uma das funções do sistema operacional é controlar todos os dispositivos de entrada e saída [I/O] do computador, emitindo comandos para os dispositivos, atendendo interrupções e manipulando erros. Deve também prover uma interface simples entre os dispositivos e o resto do sistema, e se possível, essa interface deve ser a mesma para todos os dispositivos.

O código para acesso a funções de entrada e saída representa uma fração significativa do total do sistema operacional. Todo acesso a *hardware* é obtido através do kernel do sistema, que controla os discos, impressoras e acesso à rede, entre outras coisas.

2.6 - Suporte a redes locais e distribuídas

Modernamente, a rede é quase indissociável do sistema. E se o sistema for distribuído ou do tipo sistema operacional de rede, aumenta essa dependência.

Desse modo, o kernel precisa dedicar atenção ao suporte à rede. Por exemplo, na era internet

em que vivemos praticamente todos os sistemas operacionais modernos suportam TCP/IP.

3 - Serviços do sistema

O sistema operacional precisa prover alguns serviços para os usuários e aplicações. Esses serviços variam de um sistema para outro, mas existem algumas classes de serviços que são comuns.

- Execução de programa: o sistema operacional deve ser capaz de carregar e executar aplicações. E esses processos precisam terminar normalmente ou então serem finalizados em uma situação de erro.

- Operações de I/O: o sistema precisa acessar periféricos, e isso é feito através de funções especiais. Por uma questão de eficiência e proteção, normalmente os usuários e suas aplicações não podem realizar operações de I/O diretamente.

- Manipulação do sistema de arquivos: esse serviço tem por objetivo permitir que usuários e aplicações leiam, escrevam, criem e removam arquivos.

- Comunicação: permite que um processo troque informações com outro processo. Essa interação pode ocorrer na mesma máquina ou em máquinas em rede. A comunicação pode ser implementada através de memória compartilhada ou através de troca de mensagens.

- Detecção de erros: para garantir uma computação consistente, erros na CPU, I/O, memória, programas ou hardware em geral devem ser tratados pelo sistema operacional. Isso não é garantia de tolerância a falhas, apenas detecção de erros.

- Alocação de recursos: em caso de haver múltiplos usuários e/ou múltiplos processos, o sistema operacional deve se encarregar de alocação de recursos que possam atender a todas às solicitações.

- Contabilização do uso do sistema [registro ou accounting]: consiste na manutenção do controle de quais usuários usam o sistema, e quanto de recursos computacionais eles gastam. Por exemplo, para um provedor de internet estas informações podem ser utilizadas para cobrar os serviços do usuário. Esse controle também serve para determinar estatísticas de uso, e podem ser usadas no planejamento para a compra de novos equipamentos. Normalmente todo sistema operacional moderno tem um serviço de log do sistema, que registra o que está sendo acessado pelos usuários. Por exemplo, os sistemas operacionais membros da família Unix usam o syslog.

- Auditoria e segurança do sistema [proteção]: a segurança do sistema é importante para fornecer recursos e dados somente para aqueles usuários autorizados, normalmente autenticados com uso de senha. O suporte a auditoria por parte do sistema operacional, através de logs do sistema, permite determinar se o sistema está provendo recursos e dados de forma segura. Nesse aspecto, é importante que o sistema operacional não tenha vulnerabilidade que possa ser explorada por algum atacante com o intuito de obter acesso

não autorizado.

4 - Chamadas de sistema

Na visão tradicional, as aplicações fazem solicitações ao kernel via system calls. Isso é verdade para os sistemas da família Unix. No Windows, as system calls são chamadas de API – Application Program Interface. No OpenVMS[3] é chamado de system services. Mas embora sejam conceitualmente diferentes, têm o mesmo propósito.

Uma preocupação nos projetos de sistemas operacionais é a implementação de mecanismos de proteção ao kernel e de acesso aos seus serviços. Se uma aplicação que tem acesso ao kernel realizar uma operação que altere a sua integridade, todo o sistema poderá ficar comprometido e inoperante.

As system calls podem ser entendidas como uma porta de entrada para o acesso ao kernel do sistema 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. Para cada serviço disponível existe uma system call associada e cada sistema operacional tem seu próprio conjunto de chamadas, com nomes, parâmetros e formas de ativação específicos. Isto explica por que uma aplicação desenvolvida utilizando serviços de um determinado sistema operacional não pode ser portada diretamente para outro sistema.

Os institutos ISO[4] e IEEE[5] propuseram criar uma biblioteca de chamadas padronizadas, então foi criado o padrão POSIX [Portable System Interface for Unix], que permitiu que uma aplicação desenvolvida seguindo este conjunto de chamadas pudesse ser executada em qualquer sistema operacional que oferecesse suporte a esse padrão. Atualmente, a maioria do sistemas operacionais modernos incorpora suporte a system calls POSIX.

As chamadas do sistema constituem a interface entre programas aplicativos e o sistema operacional. Essas chamadas do sistema são funções que podem ser ligadas com os aplicativos para prover serviços como leitura do relógio interno, operações de entrada/saída [Input/Output – I/O] e obter comunicação entre processos.

Através dos parâmetros fornecidos na system call, a solicitação é processada e uma resposta é retornada à aplicação juntamente com um estado de conclusão indicando se houve algum erro.

É uma forma dos programadores fazerem solicitação de serviços ao sistema operacional, similar à chamada de sub-rotinas. As chamadas de sistema transferem a execução para o sistema operacional, e o retorno dessas chamadas fazem com que a execução do programa seja retomada.

Por exemplo, a chamada:

write(fd, buffer, n_to_write); // escreve num arquivo

3 OpenVMS: Open Virtual Memory System [http://www.openvms.org/].
4 ISO: International Organization for Standardization [http://www.iso.org].
5 IEEE: Institute of Electrical and Electronics Engineers [www.ieee.org/].

na linguagem C permite escrever num arquivo previamente aberto. Além de acessar o sistema de arquivos, as chamadas de sistema também permitem o controle de processos. Por exemplo:

```
fork ( );       // cria um novo processo
exit ( );       // termina a execução do processo
kill ( );       // finaliza um processo
```

atuam em processos.

A tabela abaixo mostra um relacionamento entre chamadas de sistema e API do Windows.

Unix system calls	Windows API	Descrição
fork	CreateProcess	Cria um novo processo
exit	ExitProcess	Termina a execução do processo
open	CreateFile	Cria um arquivo novo, ou abre um existente
close	CloseHandle	Fecha um arquivo
read	ReadFile	Lê dados de um arquivo
write	WriteFile	Escreve dados num arquivo
mkdir	CreateDirectory	Cria um diretório
unlink	DeleteFile	Remove um arquivo
chdir	SetCurrentDirectory	Altera o diretório de trabalho
time	GetLocalTime	Obtém o horário local

Em linguagens de alto nível, as chamadas de sistema são encapsuladas na biblioteca do compilador. Por exemplo, *printf* ( ) não é chamada de sistema, mas sim uma rotina de biblioteca [*library routine*].

Outro aspecto interessante sobre as system calls, é que a linguagem C foi criada em 1972 para reescrever o código fonte do Unix, que era em linguagem assembly. Portanto, não é nenhuma surpresa verificar que em C exista uma função na biblioteca padrão para cada system call.

O responsável pela implementação das chamadas de sistema é o kernel.

4.1 - Programas de sistema

Os programas de sistema são algumas vezes chamados de *utilitários*, pois são programas executados fora do kernel e que implementam tarefas básicas para facilitar a utilização do sistema. Como exemplo temos uma grande variedade de utilitários para a manipulação de arquivos e pastas, que permitem listar, visualizar, criar, excluir, etc. O Windows Explorer está nessa categoria.

Por razões históricas, o programa de sistema mais importante é o *shell* ou *interpretador de comandos*, que define uma interface entre os usuários e o kernel do sistema.

4.2 - Interpretador de comandos

O interpretador de comandos é um processo que perfaz a interface do usuário com o sistema operacional. Este processo espera pelos comandos enviados via teclado [entrada padrão], interpreta e passa seus parâmetros ao kernel do sistema e depois envia o resultado do comando para a saída padrão [monitor].

À exceção dos sistemas operacionais gráficos, o *shell* é ativado sempre que o usuário inicia uma sessão. A interpretação é normalmente feita através de uma *linguagem de comandos*, porém *shells* modernos podem utilizar interfaces gráficas.

Serviços como *login* e *logout*, manipulação de arquivos e execução de programas são solicitados através do interpretador de comandos.

Alguns sistemas operacionais como Linux, Solaris, AIX, HP-UX, FreeBSD, etc., disponibilizam e permitem ao usuário escolher entre vários *shells*. A tabela abaixo mostra alguns desses shells:

Shell	Descrição
sh	Bourne shell
bash	Bourne again shell
csh	C shell
ksh	Korn shel
tcsh	C shell aprimorado
zsh	Z shell

Atualmente no Windows o interpretador de comandos é o CMD [CMD.EXE].

Versões mais antigas do Windows [Windows 3.X – ambiente operacional] eram essencialmente *shells* de substituição ao interpretador de comandos COMMAND.COM do DOS.