

INTRODUÇÃO A SISTEMAS OPERACIONAIS

**Componente: Gestão de Sistemas Operacionais I
Colégio Entec – Profª Tatiane Tolentino de Assis
tatiane@colegioentec.com.br**

2010

1 – INTRODUÇÃO

O computador é um aparelho que executa tarefas repetitivas e armazena uma grande quantidade de dados. É através de programas e/ou aplicativos, que somos capazes de interagir com o computador.

O computador basicamente é dividido em Hardware e Software.

O hardware é composto por circuitos eletrônicos (processador, memória, portas de entrada/ saída, etc) e periféricos eletro-óptico-mecânicos (teclados, mouses, discos rígidos, unidades de disquete, CD ou DVD, dispositivos USB, etc). Por sua vez, o software de aplicação é representado por programas destinados ao usuário do sistema e que constituem a razão final de seu uso, como editores de texto, navegadores Internet ou jogos. Entre os aplicativos e o hardware reside uma camada de software multifacetada e complexa, denominada genericamente de *Sistema Operacional*.

Ou seja, um S.O. é um programa ou um conjunto de programas cuja a função é servir de interface entre o computador (hardware) e o usuário.

Unidades de medida

A menor unidade utilizável para representação de informações em um computador é o BIT, que assume os valores 0 ou 1. Essa representação, dita binária, está relacionada com o fato das informações serem armazenadas fisicamente no computador na forma de uma polaridade elétrica (positivo ou negativo) ou magnética (norte ou sul nos ímãs). Como um único bit é insuficiente para representar informações mais complexas, eles são agrupados e combinados. Num primeiro agrupamento, eles são reunidos em conjuntos de oito, recebendo a denominação de BYTE (8 bits).

Quando nos referimos às informações armazenadas em um computador utilizamos, portanto, o termo byte, que corresponde a um caractere. Tendo em vista que a unidade byte é consideravelmente pequena quando indicamos valores mais extensos, utilizamos múltiplos do byte: kilobyte, megabyte, gigabyte, terabyte, etc.

1 bit = sinal elétrico
8 bits = 1 byte (1 caractere)
1 kilobyte (KB)=1024 bytes
1 megabyte (MB)=1024 kilobytes
1 gigabyte (GB)=1024 megabytes
1 terabyte (TB) = 1024 gigabytes
1 petabyte (PB) = 1024 terabytes
1 exabyte (EB) = 1024 petabytes
1 zettabyte (ZB) = 1024 exabytes
1 yottabyte (YB) = 1024 zettabytes

História dos sistemas operacionais

No início da computação os primeiros "sistemas operacionais" eram únicos, pois cada mainframe (computadores de grande porte, que eram do tamanho de um armário ou maior) vendido necessitava de um sistema operacional específico. Esse problema era resultado de arquiteturas diferentes e da linguagem utilizada — no caso, assembly (linguagem de baixo nível). Após essa fase, iniciou-se a pesquisa de sistemas operacionais que automatizassem a troca de tarefas (jobs), pois os sistemas eram mono-usuário e tinham cartões perfurados como entrada (eliminando, assim, o trabalho de pessoas que eram contratadas apenas para trocar os cartões perfurados).

Um dos primeiros sistemas operacionais de propósito geral foi o CTSS, desenvolvido no MIT. Após o CTSS, o MIT, os laboratórios Bell da AT&T e a General Electric desenvolveram o Multics, cujo objetivo era suportar centenas de usuários. Apesar do fracasso comercial, o Multics serviu como base para o estudo e desenvolvimento de sistemas operacionais. Um dos desenvolvedores do Multics, que trabalhava para a Bell, Ken Thompson, começou a rescrever o Multics num conceito menos ambicioso, criando o Unics (em 1969), que mais tarde passou a chamar-se Unix. Os sistemas operacionais eram geralmente programados em assembly, até mesmo o Unix em seu início. Então, Dennis Ritchie (também da Bell) criou a linguagem C a partir da linguagem B, que havia sido criada por Thompson. Finalmente, Thompson e Ritchie reescreveram o Unix em C. O Unix criou um ecossistema de versões, onde destacam-se: System V e derivados (HP-UX, AIX); família BSD (FreeBSD, NetBSD, OpenBSD, etc.), Linux e até o Mac OS X (que deriva do Mach e FreeBSD).

Na década de 1970, quando começaram a aparecer os computadores pessoais, houve a necessidade de um sistema operacional de utilização mais fácil. Em 1980, William (Bill) Gates e seu colega de faculdade, Paul Allen, fundadores da Microsoft, compram o sistema QDOS ("Quick and Dirty Operating System") de Tim Paterson por \$50.000, batizam-no de DOS (Disk Operating System) e vendem licenças à IBM. O DOS vendeu muitas cópias, como o sistema operacional padrão para os computadores pessoais desenvolvidos pela IBM.

Em meados de 1993, o Windows começa a ser tecnicamente considerado como sistema operacional, com o lançamento do Windows NT. O Windows 3.11, Windows 95 e Windows 98 são softwares de interface gráfica.

No começo da década de 1990, um estudante de computação finlandês postou um comentário numa lista de discussão da Usenet dizendo que estava desenvolvendo um kernel de sistema operacional e perguntou se alguém gostaria de auxiliá-lo na tarefa. Este estudante chamava-se Linus Torvalds e o primeiro passo em direção ao tão conhecido Linux foi dado naquele momento.

1.1 Objetivos

Existe uma grande distancia entre os circuitos eletrônicos e dispositivos de hardware e os programas aplicativos em software. Os circuitos são complexos, acessados através de interfaces de baixo nível (geralmente usando as portas de entrada/saída do processador) e muitas vezes suas características e seu comportamento dependem da tecnologia usada em sua construção. Por exemplo, a forma de acesso de baixo nível a discos rígidos IDE difere da forma de acesso a leitores de CD. Essa grande diversidade pode ser uma fonte de dores de cabeça para o desenvolvedor de aplicativos. Portanto, torna-se desejável oferecer aos programas aplicativos uma forma de acesso homogênea aos dispositivos físicos, que permita abstrair as diferenças tecnológicas entre eles.

O sistema operacional é uma camada de software que opera entre o hardware e os programas aplicativos voltados ao usuário final. O sistema operacional é uma estrutura de software ampla, muitas vezes complexa, que incorpora aspectos de baixo nível (como drivers de dispositivos e gerencia de memória física) e de alto nível (como programas utilitários e a própria interface gráfica).

A figura 1 ilustra a arquitetura geral de um sistema de computação típico. Nela, podemos observar elementos de hardware, o sistema operacional e alguns programas aplicativos.

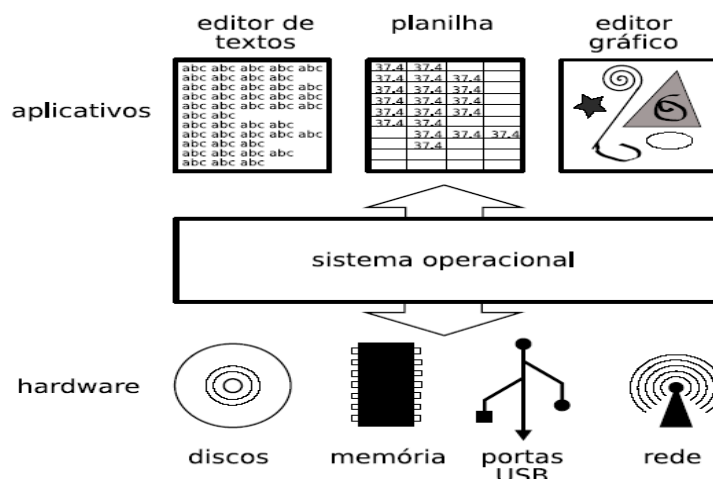


Figura 1 -Objetivos Básicos de um S.O.

1.1.1- Abstração de recursos

O acesso aos recursos de hardware de um sistema de computação pode ser trabalhoso e complicado, devido às características específicas de cada dispositivo físico e a complexidade de sua interface. Por exemplo, a seqüência a seguir apresenta os principais passos envolvidos na abertura de um arquivo (operação open) em um leitor de disquete:

1. verificar se os parâmetros informados estão o corretos (nome do arquivo, identificador do leitor de disquete, buffer de leitura, etc);
2. verificar se o leitor de disquetes esta disponível;
3. verificar se o leitor contém um disquete;
4. ligar o motor do leitor e aguardar atingir a velocidade de rotação correta;

5. posicionar a cabeça de leitura sobre a trilha onde esta a tabela de diretório;
6. ler a tabela de diretório e localizar o arquivo ou subdiretório desejado;
7. mover a cabeça de leitura para a posição do bloco inicial do arquivo;
8. ler o bloco inicial do arquivo e depositá-lo em um buffer de memória.

Assim, o sistema operacional deve definir interfaces abstratas para os recursos do hardware, visando atender os seguintes objetivos:

- **Prover interfaces de acesso aos dispositivos, mais simples de usar que as interfaces de baixo nível**, para simplificar a construção de programas aplicativos. Por exemplo: para ler dados de um disco rígido, uma aplicação usa uma abstração chamada *arquivo*, acessível através de operações como open, read e close. Caso tivesse de acessar o disco diretamente, teria de manipular portas de entrada/saída e registradores com comandos para a controladora de disco (sem falar na dificuldade de localizar os dados desejados dentro do disco).
- **Tornar os aplicativos independentes do hardware**. Ao definir uma interface abstrata de acesso a um dispositivo de hardware, o sistema operacional desacopla o hardware dos aplicativos e permite que ambos evoluam de forma mais autônoma. Por exemplo, o código de um editor de textos não deve ser dependente da tecnologia de discos rígidos utilizada no sistema.
- **Definir interfaces de acesso homogêneas para dispositivos com tecnologias distintas**. Através de suas abstrações, o sistema operacional permite aos aplicativos usar a mesma interface para dispositivos diversos. Por exemplo, um aplicativo acessa dados em disco usando a abstração de arquivo, sem levar em conta onde estão os dados reais: num disquete, num disco IDE, num disco SCSI, numa máquina fotográfica digital conectada a uma porta USB, num CD ou num disco remoto compartilhado através da rede.

1.1.2 Gerência de recursos

Os programas aplicativos usam o hardware para atingir seus objetivos: ler e armazenar dados, editar e imprimir documentos, navegar na Internet, tocar música, etc.

Em um sistema com várias atividades simultâneas, podem surgir conflitos no uso do hardware, quando dois ou mais aplicativos precisam dos mesmos recursos para poder executar. Cabe ao sistema operacional definir *políticas* para gerenciar o uso dos recursos de hardware pelos aplicativos, e resolver eventuais disputas e conflitos. Vejamos algumas situações onde a gerência de dos recursos do hardware se faz necessária:

- Cada computador possui normalmente um s o processador. O uso desse processador deve ser distribuído entre os aplicativos presentes no sistema, de forma que cada um deles possa executar na velocidade adequada para cumprir suas funções sem prejudicar os outros. O mesmo ocorre com a memória RAM, que deve ser distribuída de forma justa entre as aplicações.
- A impressor é um recurso cujo acesso deve ser efetuado de forma mutuamente exclusiva (apenas um aplicativo por vez), para não ocorrer mistura de conteúdo. Nos documentos impressos. O sistema operacional resolve essa questão definindo uma fila de trabalhos a imprimir (*print jobs*) normalmente atendidos de forma sequencial (FIFO).
- Ataques de negação de serviço (*DoS – Denial of Service*) são comuns na Internet. Eles consistem em usar diversas técnicas para forçar um servidor de rede a dedicar seus recursos a atender um determinado usuário, em detrimento dos demais. Por exemplo, ao abrir 10.000

conexões simultâneas em um servidor de e-mail POP3, um atacante pode puxar para si todos os recursos do servidor (processos, conexões de rede, memória e processador), fazendo com que os demais usuários não sejam mais atendidos. Cabe ao sistema operacional do servidor detectar tais situações e impedir que todos os recursos do sistema sejam monopolizados por um só usuário (ou um pequeno grupo).

Assim, um sistema operacional visa abstrair o hardware e gerenciar seus recursos, provendo aos aplicativos um ambiente de execução abstrato, no qual o acesso aos recursos de hardware se faz através de interfaces simples, independentes das características de baixo nível do hardware, e no qual os conflitos no uso do hardware são minimizados.

2- Tipos de sistemas operacionais

Os sistemas operacionais podem ser classificados segundo diversos parâmetros e perspectivas, como tamanho, velocidade, suporte a recursos específicos, acesso à rede, etc. A seguir são apresentados alguns tipos de sistemas operacionais usuais (muitos sistemas operacionais se encaixam bem em mais de uma das categorias apresentadas):

- **Batch (de lote):** os sistemas operacionais mais antigos trabalhavam “por lote, ou seja, todos os programas a executar eram colocados em uma fila, com seus dados e demais informações para a execução. O processador recebia um programa após o outro, processando-os em seqüência, o que permitia um alto grau de utilização do sistema. Ainda hoje o termo “em lote é usado para designar um conjunto de comandos que deve ser executado em seqüência, sem interferência do usuário. Exemplos desses sistemas incluem o OS/360 e VMS, entre outros.
- **De rede:** um sistema operacional de rede deve possuir suporte “A operação em rede, ou seja, a capacidade de oferecer as aplicações locais recursos que estejam localizados em outros computadores da rede, como arquivos e impressoras. Ele também deve disponibilizar seus recursos locais aos demais computadores, de forma controlada”. A maioria dos sistemas operacionais atuais oferece esse tipo de funcionalidade.
- **Distribuído:** em um sistema operacional distribuído, os recursos de cada máquina estão disponíveis globalmente, de forma transparente aos usuários. Ao lançar uma aplicação, o usuário interage com sua janela, mas não sabe onde ela está executando ou armazenando seus arquivos: o sistema é quem decide, de forma transparente. Os sistemas operacionais distribuídos já existem há tempos (Amoeba [TKvRB91] e Clouds [DRJLAR91], por exemplo), mas ainda não são uma realidade de mercado.
- **Multi-usuário:** Um sistema operacional multi-usuário deve suportar a identificação do “dono” de cada recurso dentro do sistema (arquivos, processos, áreas de memória, conexões de rede) e impor regras de controle de acesso para impedir o uso desses recursos por usuários não autorizados. Essa funcionalidade é fundamental para a segurança dos sistemas operacionais de rede e distribuídos. Grande parte dos sistemas atuais são multi-usuários.
- **Desktop:** um sistema operacional “de mesa” é voltado ao atendimento do usuário doméstico e corporativo para a realização de atividades corriqueiras, como edição de textos e gráficos, navegação na Internet e reprodução de mídias simples. Suas principais características são a interface gráfica, o suporte a interatividade e a

operação em rede. Exemplos de sistemas *desktop* são o Windows XP, MacOS X e Linux.

- **Servidor:** um sistema operacional servidor deve permitir a gestão eficiente de grandes quantidades de recursos (disco, memória, processadores), impondo prioridades e limites sobre o uso dos recursos pelos usuários e seus aplicativos. Normalmente um sistema operacional servidor também tem suporte a rede e multi-usuários.
- **Embutido:** um sistema operacional é dito embutido (*embedded*) quando é construído para operar sobre um hardware com poucos recursos de processamento, armazenamento e energia. Aplicações típicas desse tipo de sistema aparecem em telefones celulares, controladores industriais e automotivos, equipamentos eletrônicos de uso doméstico (leitores de DVD, TVs, fornos-micro-ondas, centrais de alarme, etc). Muitas vezes um sistema operacional embutido de apresenta na forma de uma biblioteca a ser ligada ao programa da aplicação (que é fixa). Exemplos de sistemas operacionais embutidos são o μ C/OS, Xylinx, LynxOS e VxWorks.
- **Tempo real:** ao contrario da concepção usual, um sistema operacional de tempo real não precisa ser necessariamente ultra-rápido; sua característica essencial é ter um comportamento temporal previsível (ou seja, seu tempo de resposta deve ser conhecido no melhor e pior caso de operação). A estrutura interna de um sistema operacional de tempo real deve ser construída de forma a minimizar esperas e latências imprevisíveis, como tempos de acesso a disco e sincronizações excessivas. Existem duas classificações de sistemas de tempo real: *soft real-time systems*, nos quais a perda de prazos implica na degradação do serviço prestado. Um exemplo seria o suporte a gravação de CDs ou à reprodução de músicas. Caso o sistema se atrase, pode ocorrer à perda da mídia em gravação ou falhas na musica que está sendo tocada. Por outro lado, nos *hard real-time systems* a perda de prazos pelo sistema pode perturbar o objeto controlado, com graves conseqüências humanas, econômicas ou ambientais. Exemplos desse tipo de sistema seriam o controle de funcionamento de uma turbina de avião a jato ou de uma caldeira industrial. Exemplos de sistemas de tempo real incluem o QNX, RT-Linux e VxWorks. Muitos sistemas embutidos têm características de tempo real, e vice-versa.

3 - Funcionalidades

Para cumprir seus objetivos de abstração e gerência, o sistema operacional deve atuar em varias frentes. Cada um dos recursos do sistema possui suas particularidades, o que impõe exigências específicas para gerenciar e abstrair os mesmos. Sob essa perspectiva, as principais funcionalidades implementadas por um sistema operacional típico são:

- **Gerência do processador:** também conhecida como gerencia de processos ou de atividades, esta funcionalidade visa distribuir a capacidade de processamento de forma justa entre as aplicações, evitando que uma aplicação monopolize esse recurso e respeitando as prioridades dos usuários. Busca-se criar a abstração de “um processador para cada tarefa”, que facilita a vida dos programadores de aplicações e permite a construção de sistemas mais interativos. Também faz parte da gerencia de

atividades fornecer abstrações para sincronizar atividades interdependentes e prover formas de comunicação entre elas.

- **Gerência de memória:** tem como objetivo fornecer a cada aplicação um espaço de memória próprio, independente e isolado dos demais, inclusive do núcleo do sistema. Caso a memória RAM não seja suficiente, o sistema deve prover armazenamento secundário (espaço em disco) como complemento de memória, de forma transparente as aplicações. A principal abstração construída pela gerência de memória é a noção de *memória virtual*, que desvincula o espaço de endereços visto por cada aplicação dos respectivos espaços de armazenamento providos pela RAM e pelos discos. Com isso, os programadores podem construir suas aplicações sem se preocupar com os endereços de memória onde elas irão executar.
- **Gerência de dispositivos:** cada periférico do computador possui suas peculiaridades; assim, o procedimento de interação com uma placa de rede é completamente diferente da interação com um disco rígido SCSI. Todavia, existem muitos problemas e abordagens em comum para o acesso aos periféricos. Por exemplo, é possível criar uma abstração única para a maioria dos dispositivos de armazenamento como *pen-drives*, discos SCSI ou IDE, disquetes, etc, na forma de um vetor de blocos de dados. A função da gerência de dispositivos (também conhecida como *gerência de entrada/saída*) é implementar a interação com cada dispositivo por meio de *drivers* e criar modelos abstratos que permitam agrupar vários dispositivos distintos sob a mesma interface de acesso.
- **Gerência de arquivos:** esta funcionalidade é construída sobre a gerência de dispositivos e visa criar as abstrações de arquivo e diretório, definindo também sua interface de acesso e as regras para seu uso. É importante observar que essas abstrações são tão importantes e difundidas que muitos sistemas operacionais as usam para permitir o acesso a recursos que nada tem a ver com armazenamento, como conexões de rede (nos sistemas UNIX e Windows, cada socket TCP é visto como um descritor de arquivo no qual pode-se ler ou escrever dados), informações do núcleo do sistema (como o diretório `/proc` do UNIX) ou mesmo para abstrair todos os recursos do sistema (como faz o sistema operacional *Plan 9* [PPT+93], para o qual todos os recursos são vistos como arquivos).
- **Gerência de proteção:** com computadores conectados em rede e compartilhados por vários usuários, é importante definir claramente os recursos que cada usuário pode acessar, as formas de acesso permitidas (leitura, escrita, etc) e garantir que essas definições serão cumpridas. Para proteger os recursos do sistema contra acessos indevidos, é necessário: a) definir usuários e grupos de usuários; b) identificar os usuários que se conectam ao sistema, através de procedimentos de autenticação; c) definir e aplicar regras de controle de acesso aos recursos, relacionando todos os usuários, recursos e formas de acesso e aplicando essas regras através de procedimentos de autorização; e finalmente d) registrar o uso dos recursos pelos usuários, para fins de auditoria e contabilização.

Alem dessas funcionalidades básicas, oferecidas pela maioria dos sistemas operacionais, várias outras vêm se agregar aos sistemas modernos, para cobrir aspectos complementares, como a interface gráfica, suporte de rede, fluxos multimídia, gerência de energia, etc.

As funcionalidades do sistema operacional geralmente são interdependentes: por exemplo, a gerência do processador depende de aspectos da gerência de memória, assim como a gerência de memória depende da gerência de dispositivos e da gerência de proteção. Alguns autores representam a estrutura do sistema operacional conforme indicado na figura 2. Nela, o núcleo central implementa o acesso de baixo nível ao hardware, enquanto os módulos externos representam as várias funcionalidades do sistema.

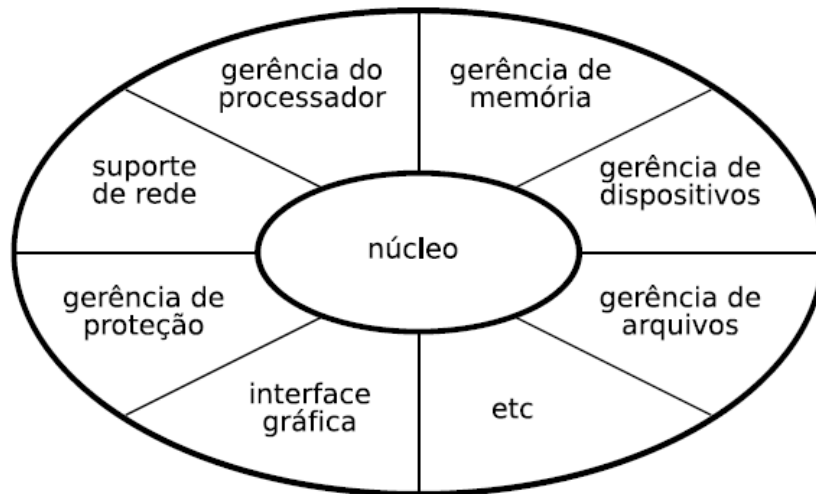


Figura 2: Funcionalidades do sistema operacional

Uma regra importante a ser observada na construção de um sistema operacional é separação entre os conceitos de política e mecanismo. Como *política* consideram-se os aspectos de decisão mais abstratos, que podem ser resolvidos por algoritmos de nível mais alto, como, por exemplo, decidir a quantidade de memória que cada aplicação ativa deve receber, ou qual o próximo pacote de rede a enviar para satisfazer determinadas especificações de qualidade de serviço. Por outro lado, como *mecanismo* consideram-se os procedimentos de baixo nível usados para implementar as políticas, ou seja, atribuir ou retirar memória de uma aplicação, enviar ou receber um pacote de rede, etc. Os mecanismos devem ser suficientemente genéricos para suportar mudanças de política sem necessidade de modificações. Essa separação entre os conceitos de política e mecanismo traz uma grande flexibilidade aos sistemas operacionais, permitindo alterar sua personalidade (sistemas mais interativos ou mais eficientes, etc) sem ter de mexer no código que interage diretamente com o hardware. Alguns sistemas, como o InfoKernel, permitem as aplicações escolher as políticas do sistema mais adequadas para suas necessidades.

4 - Estrutura de um sistema operacional

Um sistema operacional não é um bloco único e fechado de software executando sobre o hardware. Na verdade, ele é composto de diversos componentes com objetivos e funcionalidades complementares. Alguns dos componentes mais relevantes de um sistema operacional típico são:

- **Núcleo:** é o coração do sistema operacional, responsável pela gerencia dos recursos do hardware usados pelas aplicações. Ele também implementa as principais abstrações utilizadas pelos programas aplicativos.
- **Drivers:** módulos de código específicos para acessar os dispositivos físicos. Existe um driver para cada tipo de dispositivo, como discos rígidos IDE, SCSI, portas USB, placas de vídeo, etc. Muitas vezes o driver é construído pelo próprio fabricante do hardware e fornecido em forma binária para ser acoplado ao restante do sistema operacional.
- **Código de inicialização:** a inicialização do hardware requer uma serie de tarefas complexas, como reconhecer os dispositivos instalados, testá-los e configurá-los adequadamente para seu uso posterior. Outra tarefa importante é carregar o núcleo do sistema operacional em memória e iniciar sua execução.
- **Programas utilitários:** são programas que facilitam o uso do sistema computacional, fornecendo funcionalidades complementares ao núcleo, como formatação de discos e mídias, configuração de dispositivos, manipulação de arquivos (mover, copiar, apagar), interpretador de comandos, terminal, interface gráfica, gerencia de janelas, etc.

5 - Proteção do núcleo

Um sistema operacional deve gerenciar os recursos do hardware, fornecendo-os as aplicações conforme suas necessidades. Para assegurar a integridade dessa gerencia, é essencial garantir que as aplicações não consigam acessar o hardware diretamente, mas sempre através de pedidos ao sistema operacional, que avalia e intermédia todos os acessos ao hardware. Mas como impedir as aplicações de acessar o hardware diretamente?

Núcleo, drivers, utilitários e aplicações são constituídos basicamente de código de maquina. Todavia, devem ser diferenciados em sua capacidade de interagir como hardware: enquanto o núcleo e os drivers devem ter pleno acesso ao hardware (para poder configurá-lo e gerenciá-lo), os utilitários e os aplicativos devem ter acesso mais restrito a ele, para não interferir nas configurações e na gerencia, o que acabaria desestabilizando o sistema inteiro. Além disso, aplicações com acesso pleno ao hardware tornariam inúteis os mecanismos de segurança e controle de acesso aos recursos (arquivos, diretórios, áreas de memória, etc).

Para permitir a diferenciação de privilegio de acesso entre os diferentes tipos de software, os processadores modernos contam com dois ou mais *níveis de privilegio de execução*. Esses níveis são controlados por flags especiais nos processadores, e a mudança de um nível de execução para outro é controlada por condições específicas. O processador Pentium, por exemplo, conta com 4 níveis de privilegio (sendo 0 o nível mais privilegiado), embora a maioria dos sistemas operacionais construídos para esse processador use os níveis extremos (0 para o núcleo e drivers do sistema operacional e 3 para utilitários e aplicações). Na forma mais simples desse esquema, podemos considerar dois níveis básicos de privilegio:

Nível núcleo: também denominado nível *supervisor*, *sistema*, *monitor* ou ainda *kernel space*. Para um código executando nesse nível, todo o processador esta acessível: todos os registradores, portas de entrada/saída e áreas de memória podem ser acessados em leitura e escrita. Além disso, todas as instruções do processador podem ser executadas. Ao ser ligado, o processador entra em operação neste nível.

Nível usuário (ou *userspace*): neste nível, somente um sub-conjunto das instruções do processador, registradores e portas de entrada/saída estão disponíveis. Instruções “perigosas” como HALT (parar o processador) e RESET (reiniciar o processador) são proibidas para todo código executando neste nível. Além disso, o hardware restringe o uso da memória, permitindo o acesso somente a áreas previamente definidas. Caso o código em execução tente executar uma instrução proibida ou acessar uma área de memória inacessível, o hardware irá gerar uma exceção, desviando a execução para uma rotina de tratamento dentro do núcleo, que provavelmente irá abortar o programa em execução (e também gerar a famosa frase “este programa executou uma instrução ilegal e será finalizado”, no caso do Windows).

É fácil perceber que, em um sistema operacional convencional, o núcleo e os drivers operam no nível núcleo, enquanto os utilitários e as aplicações operando nível usuário, confinados em áreas de memória distintas, conforme ilustrado na figura 3

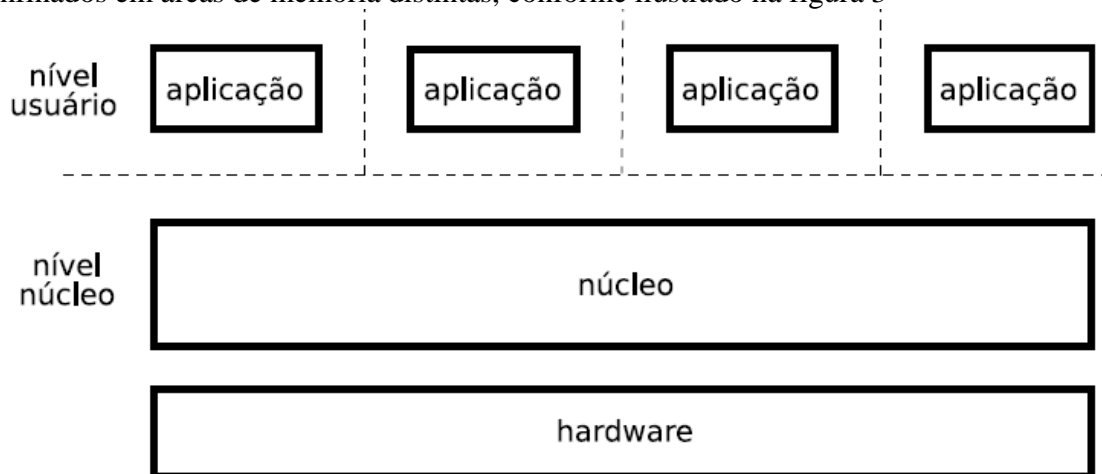


Figura 3 – Separação entre núcleos e Aplicações

6 - Arquiteturas de Sistemas Operacionais

Embora a definição de níveis de privilégio imponha uma estruturação mínima a um sistema operacional, as múltiplas partes que compõem o sistema podem ser organizadas de diversas formas, separando suas funcionalidades e modularizando.

Seu projeto. Nesta seção serão apresentadas as arquiteturas mais populares para a organização de sistemas operacionais.

6.1 Sistemas monolíticos

Em um sistema monolítico, todos os componentes do núcleo operam em modo núcleo e se inter-relacionam conforme suas necessidades, sem restrições de acesso entre si (pois o código no nível núcleo tem acesso pleno a todos os recursos e áreas de memória). A figura 4 ilustra essa arquitetura.

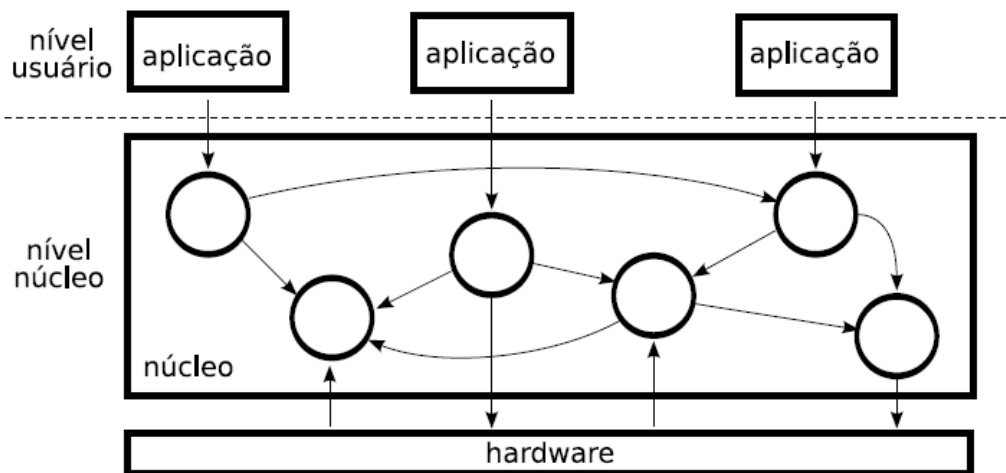


Figura 4: Uma arquitetura monolítica

A grande vantagem dessa arquitetura é seu desempenho: qualquer componente do núcleo pode acessar os demais componentes, toda a memória ou mesmo dispositivos periféricos diretamente, pois não há barreiras impedindo esse acesso. A interação direta entre componentes também leva a sistemas mais compactos.

Todavia, a arquitetura monolítica pode pagar um preço elevado por seu desempenho: a robustez e a facilidade de desenvolvimento. Caso um componente do núcleo perca o controle devido a algum erro, esse problema pode se alastrar rapidamente por todo o núcleo, levando o sistema ao colapso (travamento, reinicialização ou funcionamento errático). Além disso, a manutenção e evolução do núcleo se tornam mais complexas, porque as dependências e pontos de interação entre os componentes podem não ser evidentes: pequenas alterações na estrutura de dados de um componente podem ter um impacto inesperado em outros componentes, caso estes acessem aquela estrutura diretamente.

A arquitetura monolítica foi a primeira forma de organizar os sistemas operacionais; sistemas UNIX antigos e o MS-DOS seguiam esse modelo. Atualmente, apenas sistemas operacionais embutidos usam essa arquitetura, devido às limitações do hardware sobre o qual executam. O núcleo do Linux nasceu monolítico, mas vem sendo paulatinamente estruturado e modularizado desde a versão 2.0 (embora boa parte de seu código ainda permaneça no nível de núcleo).

6.2 Sistemas em camadas

Uma forma mais elegante de estruturar um sistema operacional faz uso da noção de camadas: a camada mais baixa realiza a interface como hardware, enquanto as camadas intermediárias provêm níveis de abstração e gerencia cada vez mais sofisticados. Por fim, a camada superior define a interface do núcleo para as aplicações (as chamadas de sistema). Essa abordagem de estruturação de software fez muito sucesso no domínio das redes de computadores, através do modelo de referência OSI (*Open Systems Interconnection*), e

também seria de se esperar sua adoção no domínio dos sistemas operacionais. No entanto, alguns inconvenientes limitam sua aceitação nesse contexto:

- O empilhamento de varias camadas de software faz com que cada pedido de uma aplicação demore mais tempo para chegar ate o dispositivo periférico ou recurso a ser acessado, prejudicando o desempenho do sistema.
- Não é obvio como dividir as funcionalidades de um núcleo de sistema operacional em camadas horizontais de abstração crescente, pois essas funcionalidades são inter-dependentes, embora tratem muitas vezes de recursos distintos.

Em decorrência desses inconvenientes, a estruturação em camadas É apenas parcialmente adotada hoje em dia. Muitos sistemas implementam uma camada inferior de abstração do hardware para interagir com os dispositivos (a camada *HAL – Hardware Abstraction Layer*, implementada no Windows NT e seus sucessores), e também organizam em camadas alguns sub-sistemas como a gerência de arquivos e o suporte de rede (seguindo o modelo OSI). Como exemplos de sistemas fortemente estruturados em camadas podem ser citados o IBM OS/2 e o MULTICS.

6.3 Sistemas micronúcleo

Uma outra possibilidade de estruturação consiste em retirar do núcleo todo o código de alto nível (normalmente associado às políticas de gerencia de recursos), deixando no núcleo somente o código de baixo nível necessário para interagir como hardware e criar as abstrações fundamentais (como a noção de atividade). Por exemplo, usando essa abordagem o código de acesso aos blocos de um disco rígido seria mantido no núcleo, enquanto as abstrações de arquivo e diretório seriam criadas e mantidas por um código fora do núcleo, executando da mesma forma que uma aplicação do usuário.

Por fazer os núcleos de sistema ficarem menores, essa abordagem foi denominada *micronúcleo* (ou *kernel*). Um micronúcleo normalmente implementa somente a noção de atividade, de espaços de memória protegidos e de comunicação entre atividades. Todos os aspectos de alto nível, como políticas de uso do processador e da memória, o sistema de arquivos e o controle de acesso aos recursos são implementados fora do núcleo, em processos que se comunicam usando as primitivas do núcleo. A figura 5 ilustra essa abordagem.

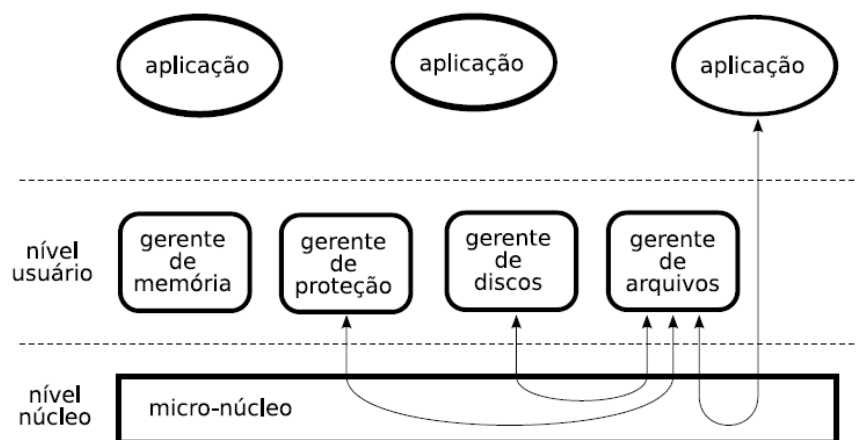


Figura 9: Visão geral de uma arquitetura micronúcleo

Em um sistema micro-núcleo, as interações entre componentes e aplicações são feitas através de trocas de mensagens. Assim, se uma aplicação deseja abrir um arquivo no disco rígido, envia uma mensagem para o gerente de arquivos, que por sua vez se comunica com o gerente de dispositivos para obter os blocos de dados relativos ao arquivo desejado. Todas as mensagens são transmitidas através de serviços do micronúcleo, como mostra a figura 9. Como os processos tem de solicitar “serviços” uns dos outros (para poder realizar suas incumbências), essa abordagem também foi denominada *cliente-servidor*.

Os micronúcleos foram muito investigados durante os anos 80. Dois exemplos clássicos dessa abordagem são os sistemas Mach e Chorus. As principais vantagens dos sistemas micronúcleo são sua robustez e flexibilidade: caso um sub-sistema tenha problemas, os mecanismos de proteção de memória e níveis de privilegio irão confiná-lo, impedindo que a instabilidade se alastre ao restante do sistema. Além disso, é possível customizar o sistema operacional, iniciando somente os 20 componentes necessários ou escolhendo os componentes mais adequados às aplicações que serão executadas.

Vários sistemas operacionais atuais adotam parcialmente essa estruturação; por exemplo, o MacOSX da Apple tem suas raízes no sistema Mach, ocorrendo o mesmo com o Digital UNIX. Todavia, o custo associado às trocas de mensagens entre componentes pode ser bastante elevado, o que prejudica seu desempenho e diminui a aceitação desta abordagem. O QNX é um dos poucos exemplos de micronúcleo amplamente utilizado, sobretudo em sistemas embutidos e de tempo-real.