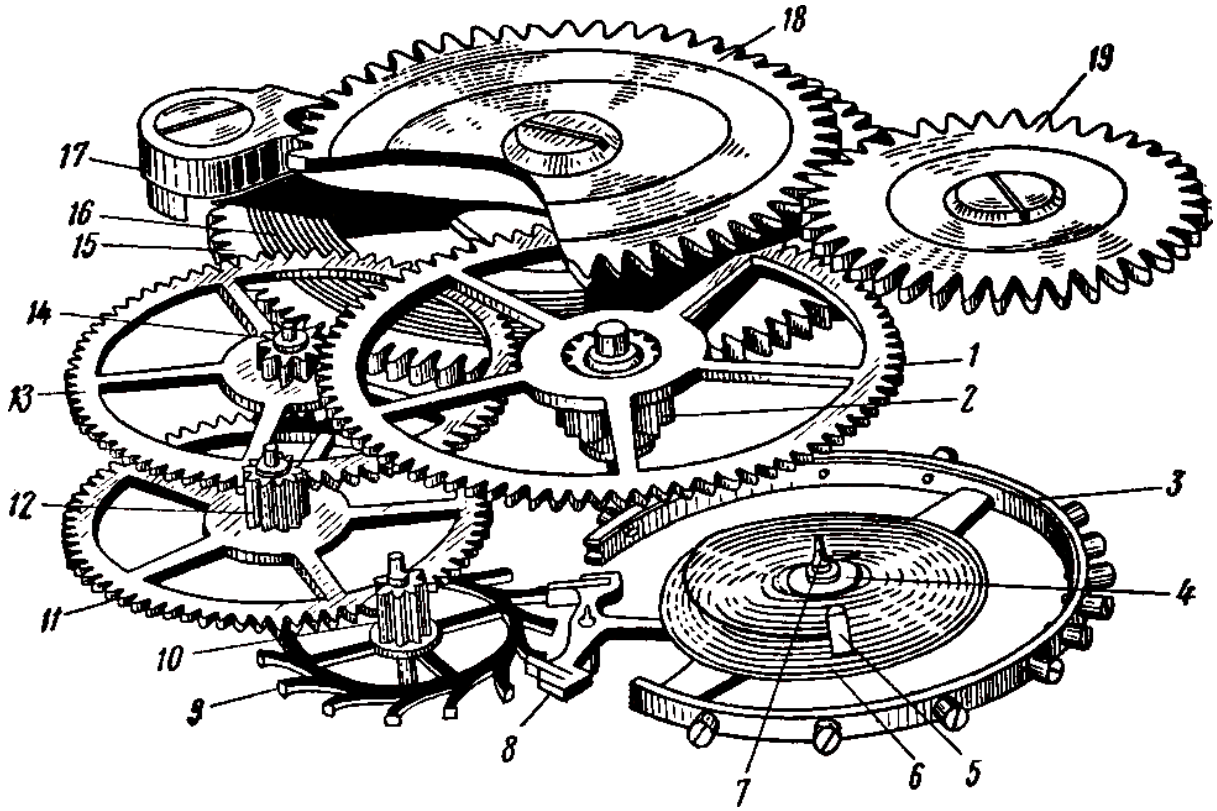
Sistemas Operacionais: Conceitos e Mecanismos

Prof. Carlos A. Maziero



UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Capítulo 3

Arquiteturas de SOs

Embora a definição de níveis de privilégio (Seção 2.3) imponha uma estruturação mínima a um sistema operacional, forçando a separação entre o núcleo e o espaço de usuário, os vários elementos que compõem o sistema podem ser organizados de diversas formas, separando suas funcionalidades e modularizando seu projeto. Neste capítulo serão apresentadas as arquiteturas mais populares para a organização de sistemas operacionais.

3.1 Sistemas monolíticos

Em um sistema monolítico[1], o sistema operacional é um "bloco maciço" de código que opera em modo núcleo, com acesso a todos os recursos do hardware e sem restrições de acesso à memória. Por isso, os componentes internos do sistema operacional podem se relacionar entre si conforme suas necessidades. A Figura 3.1 ilustra essa arquitetura.

A grande vantagem da arquitetura monolítica é seu desempenho: qualquer componente do núcleo pode acessar os demais componentes, áreas de memória ou mesmo dispositivos periféricos diretamente, pois não há barreiras impedindo esses acessos. A interação direta entre componentes leva a sistemas mais rápidos e compactos, pois não há necessidade de utilizar mecanismos específicos de comunicação entre os componentes do núcleo.

Todavia, a arquitetura monolítica pode levar a problemas de robustez do sistema. Como todos os componentes do SO têm acesso privilegiado ao hardware, caso um componente perca o controle devido a algum erro (acessando um ponteiro inválido ou uma posição inexistente em um vetor, por exemplo), esse erro pode se propagar rapidamente por todo o núcleo, levando o sistema ao colapso (travamento, reinicialização ou funcionamento errático).

Outro problema relacionado às arquiteturas monolíticas diz respeito ao processo de desenvolvimento. Como os componentes do sistema têm acesso direto uns aos outros, podem ser fortemente interdependentes, tornando a manutenção e evolução do núcleo mais complexas. Como as dependências e pontos de interação entre os componentes podem ser pouco evidentes, pequenas alterações na estrutura de dados de um componente podem ter um impacto inesperado em outros componentes, caso estes acessem aquela estrutura diretamente.

[1] A palavra "monólito" vem do grego *monos* (único ou unitário) e *lithos* (pedra).

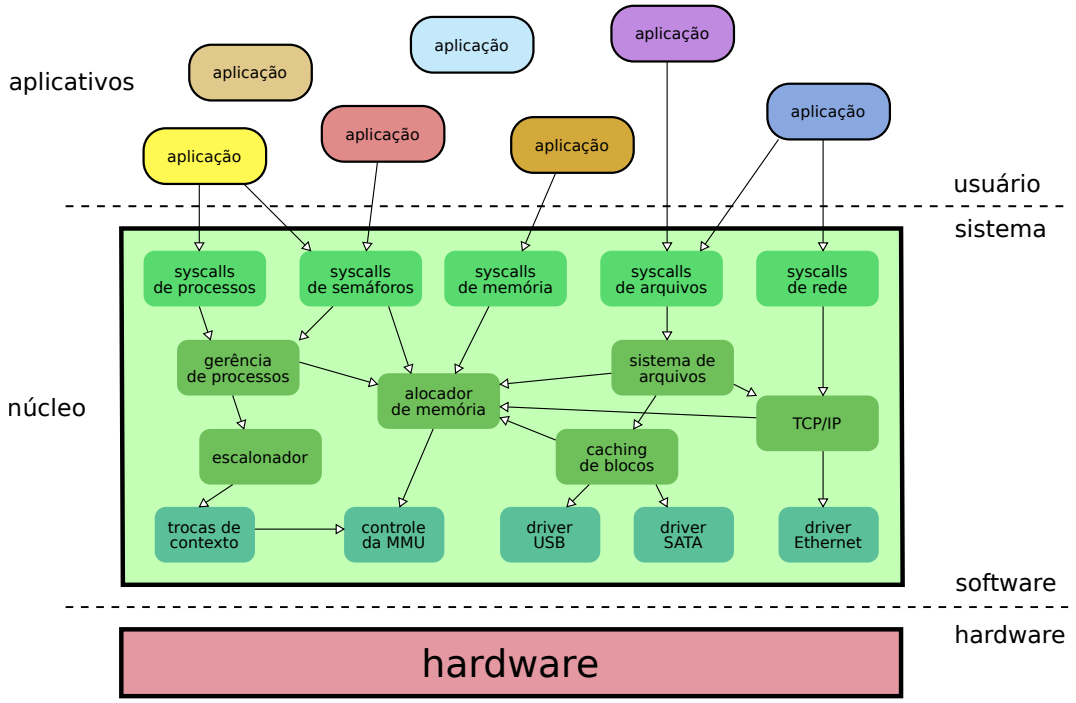


Figura 3.1: Núcleo de sistema operacional monolítico.

A arquitetura monolítica foi a primeira forma de organizar os sistemas operacionais; sistemas UNIX antigos e o MS-DOS seguiam esse modelo. O núcleo do Linux é monolítico, mas seu código vem sendo gradativamente estruturado e modularizado desde a versão 2.0 (embora boa parte do código ainda permaneça no nível de núcleo). O sistema FreeBSD [McKusick and Neville-Neil, 2005] também usa um núcleo monolítico.

3.2 Sistemas micronúcleo

Outra possibilidade de estruturar o SO consiste em retirar do núcleo todo o código de alto nível, normalmente associado às abstrações de recursos, deixando no núcleo somente o código de baixo nível necessário para interagir com o hardware e criar algumas abstrações básicas. O restante do código seria transferido para programas separados no espaço de usuário, denominados *serviços*. Por fazer o núcleo de sistema ficar menor, essa abordagem foi denominada *micronúcleo* (ou *µ-kernel*).

A abordagem micronúcleo oferece maior modularidade, pois cada serviço pode ser desenvolvido de forma independente dos demais; mais flexibilidade, pois os serviços podem ser carregados e desativados conforme a necessidade; e mais robustez, pois caso um serviço falhe, somente ele será afetado, devido ao confinamento de memória entre os serviços.

Um micronúcleo normalmente implementa somente a noção de tarefa, os espaços de memória protegidos para cada aplicação, a comunicação entre tarefas e as operações de acesso às portas de entrada/saída (para acessar os dispositivos). Todos os aspectos de alto nível, como políticas de uso do processador e da memória, o sistema de arquivos, o controle de acesso aos recursos e até mesmos os *drivers* são implementados fora do núcleo, em processos que se comunicam usando o mecanismo de comunicação provido pelo núcleo.

Um bom exemplo de sistema micronúcleo é o Minix 3 [Herder et al., 2006]. Neste sistema, o núcleo oferece funcionalidades básicas de gestão de interrupções, configuração da CPU e da MMU, acesso às portas de entrada/saída e primitivas de troca de mensagens entre aplicações. Todas as demais funcionalidades, como a gestão de arquivos e de memória, protocolos de rede, políticas de escalonamento, etc., são providas por processos fora do núcleo, denominados *servidores*. A Figura 3.2 apresenta a arquitetura do Minix 3:

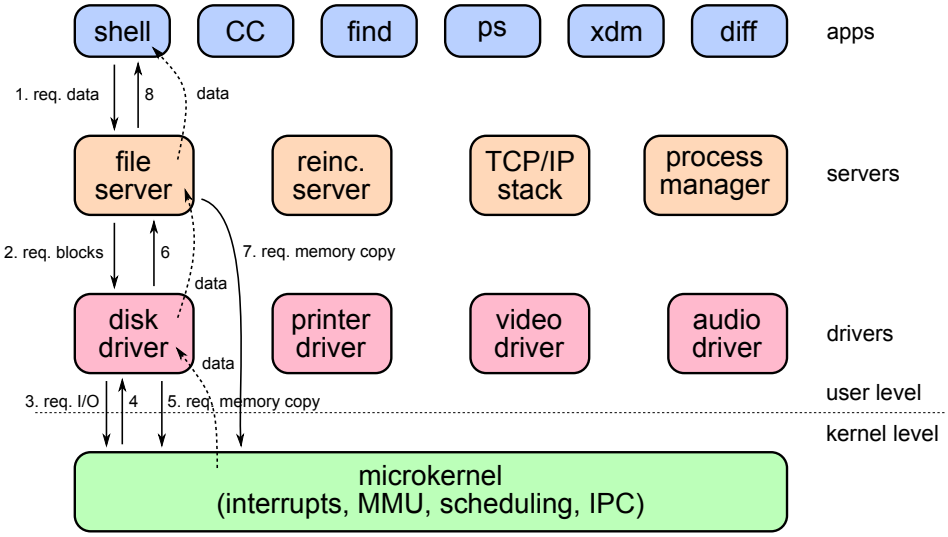


Figura 3.2: Visão geral da arquitetura do MINIX 3 (adaptado de [Herder et al., 2006]).

Por exemplo, no sistema Minix 3 a seguinte sequência de ações ocorre quando uma aplicação deseja ler dados de um arquivo no disco:

1. usando as primitivas de mensagens do núcleo, a aplicação envia uma mensagem (m_1) ao servidor de arquivos, solicitando a leitura de dados de um arquivo;

2. o servidor de arquivos verifica se possui os dados em seu cache local; se não os possuir, envia uma mensagem (m_2) ao *driver* de disco solicitando a leitura dos dados do disco em seu espaço de memória;

3. o *driver* de disco envia uma mensagem (m_3) ao núcleo solicitando operações nas portas de entrada/saída do controlador de disco, para ler dados do mesmo;

4. o núcleo verifica se o *driver* de disco tem permissão para usar as portas de entrada/saída e agenda a operação solicitada;

5. quando o disco concluir a operação, o núcleo transfere os dados lidos para a memória do *driver* de disco e responde (m_4) a este;

6. o *driver* de disco solicita (m_5) então ao núcleo a cópia dos dados recebidos para a memória do servidor de arquivos e responde (m_6) à mensagem anterior deste, informando a conclusão da operação;

7. o servidor de arquivos solicita (m_7) ao núcleo a cópia dos dados recebidos para a memória da aplicação e responde (m_8) à mensagem anterior desta;

8. a aplicação recebe a resposta de sua solicitação e usa os dados lidos.

Da sequência acima pode-se perceber que foram necessárias 8 mensagens (m_i) para realizar uma leitura de dados do disco, cada uma correspondendo a uma chamada de sistema. Cada chamada de sistema tem um custo elevado, pois implica na mudança do fluxo de execução e reconfiguração da memória acessível pela MMU. Além disso, foram necessárias várias cópias de dados entre as áreas de memória de cada entidade. Esses fatores levam a um desempenho bem inferior ao da abordagem monolítica, razão que dificulta a adoção plena dessa abordagem.

O micronúcleos vem sendo investigados desde os anos 1980. Dois exemplos clássicos dessa abordagem são os sistemas Mach [Rashid et al., 1989] e Chorus [Rozier and Martins, 1987]. Os melhores exemplos de micronúcleos bem sucedidos são provavelmente o Minix 3 [Herder et al., 2006] e o QNX. Contudo, vários sistemas operacionais atuais adotam parcialmente essa estruturação, adotando núcleos híbridos, como por exemplo o MacOS X da Apple, o Digital UNIX e o Windows NT.

3.3 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 com o hardware, enquanto as camadas intermediárias proveem níveis de abstração e gerência cada vez mais sofisticados. Por fim, a camada superior define a interface do núcleo para as aplicações (as chamadas de sistema). As camadas têm níveis de privilégio decrescentes: a camada inferior tem acesso total ao hardware, enquanto a superior tem acesso bem mais restrito (vide Seção 2.2.3).

A abordagem de estruturação de software em camadas teve sucesso no domínio das redes de computadores, através do modelo de referência OSI (*Open Systems Interconnection*) [Day, 1983], e também seria de se esperar sua adoção no domínio dos sistemas operacionais. No entanto, alguns inconvenientes limitam a aplicação do modelo em camadas de forma intensiva nos sistemas operacionais:

- O empilhamento de várias camadas de software faz com que cada pedido de uma aplicação demore mais tempo para chegar até o dispositivo periférico ou recurso a ser acessado, prejudicando o desempenho.

- Nem sempre a divisão de funcionalidades do sistema em camadas é óbvia, pois muitas dessas funcionalidades são interdependentes e não teriam como ser organizadas em camadas. Por exemplo, a gestão de entrada/saída necessita de serviços de memória para alocar/liberar *buffers* para leitura e escrita de dados, mas a gestão de memória precisa da gestão de entrada/saída para implementar a paginação em disco (*paging*). Qual dessas duas camadas viria antes?

Em decorrência desses inconvenientes, a estruturação em camadas é apenas parcialmente adotada hoje em dia. Como exemplo de sistema fortemente estruturado em camadas pode ser citado o MULTICS [Corbató and Vyssotsky, 1965].

Muitos sistemas mais recentes 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 subsistemas, como a gerência de arquivos e o suporte de rede (seguindo

o modelo OSI). Essa organização parcial em camadas pode ser facilmente observada nas arquiteturas do Minix 3 (Figura 3.2) Windows 2000 (Figura 3.3) e Android (Figura 2.2).

3.4 Sistemas híbridos

Apesar das boas propriedades de modularidade, flexibilidade e robustez proporcionadas pelos micronúcleos, sua adoção não teve o sucesso esperado devido ao baixo desempenho. Uma solução encontrada para esse problema consiste em trazer de volta ao núcleo os componentes mais críticos, para obter melhor desempenho. Essa abordagem intermediária entre o núcleo monolítico e micronúcleo é denominada *núcleo híbrido*. É comum observar também nos núcleos híbridos uma influência da arquitetura em camadas.

Vários sistemas operacionais atuais empregam núcleos híbridos, entre eles o Microsoft Windows, a partir do Windows NT. As primeiras versões do Windows NT podiam ser consideradas micronúcleo, mas a partir da versão 4 vários subsistemas foram movidos para dentro do núcleo para melhorar seu desempenho, transformando-o em um núcleo híbrido. Os sistemas MacOS e iOS da Apple também adotam um núcleo híbrido chamado XNU (de *X is Not Unix*), construído a partir dos núcleos Mach (micronúcleo) [Rashid et al., 1989] e FreeBSD (monolítico) [McKusick and Neville-Neil, 2005]. A figura 3.3 mostra a arquitetura interna do núcleo usado no SO Windows 2000.

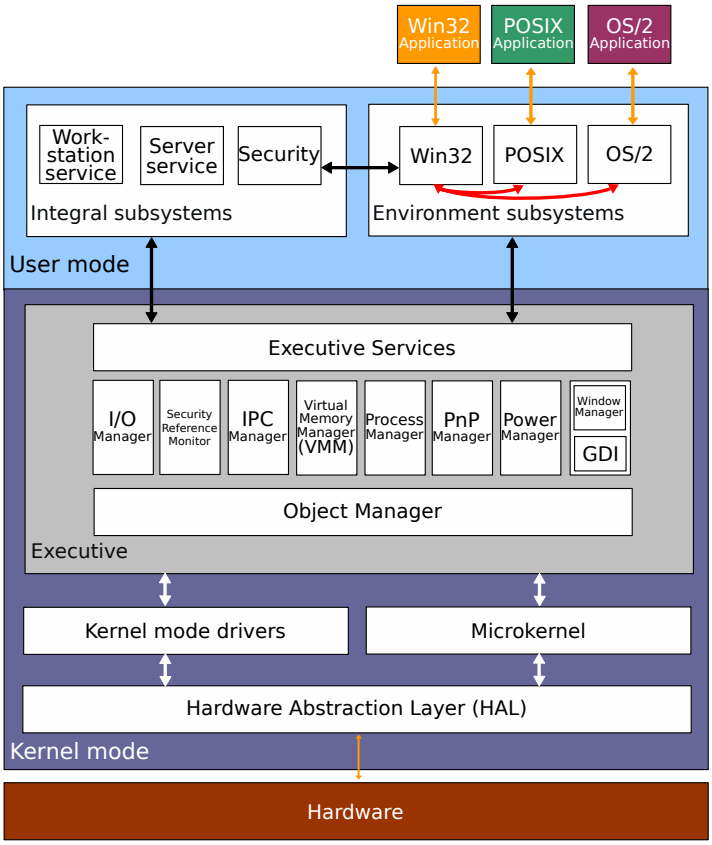


Figura 3.3: Arquitetura do sistema Windows 2000 [Wikipedia, 2018].